

Run-Time Library (RTL) :
Reference guide.

Reference guide for RTL units.
Document version 2.0
December 2004

Michaël Van Canneyt

Contents

0.1	Overview	70
1	Reference for unit 'BaseUnix'	71
1.1	Used units	71
1.2	Overview	71
1.3	Constants, types and variables	71
	Constants	71
	Types	88
1.4	Procedures and functions	99
	FpAccess	99
	FpAlarm	100
	FpChdir	100
	FpChmod	101
	FpChown	102
	FpClose	103
	FpClosedir	103
	FpDup	104
	FpDup2	104
	FpExecv	105
	FpExecve	106
	FpExit	107
	FpFcntl	108
	fpfdfillset	108
	fpFD_CLR	108
	fpFD_ISSET	109
	fpFD_SET	109
	fpFD_ZERO	109
	FpFork	109
	FpFStat	110
	FpFtruncate	111
	FpGetcwd	111

FpGetegid	111
FpGetEnv	112
fpgeterrno	112
FpGeteuid	113
FpGetgid	113
FpGetgroups	114
FpGetpgrp	114
FpGetpid	114
FpGetppid	115
fpGetPriority	115
FpGetuid	115
FpIOctl	116
FpKill	116
FpLink	117
FpLseek	118
fpLstat	119
FpMkdir	120
FpMkfifo	120
Fpmmmap	121
Fpmunmap	122
FpNanoSleep	122
fpNice	123
FpOpen	124
FpOpendir	125
FpPause	126
FpPipe	127
FpRead	127
FpReaddir	129
fpReadLink	129
FpRename	130
FpRmdir	131
fpSelect	131
fpseterrno	132
FpSetgid	133
fpSetPriority	133
FpSetsid	134
fpsettimeofday	134
FpSetuid	134
FPSigaction	134
FpSigAddSet	135

FpSigDelSet	136
FpSigEmptySet	136
FpSigFillSet	136
FpSigIsMember	137
FpSignal	137
FpSigPending	138
FpSigProcMask	138
FpSigSuspend	138
FpSleep	139
FpStat	139
fpSymlink	140
fpS_ISBLK	141
fpS_ISCHR	142
fpS_ISDIR	142
fpS_ISFIFO	142
fpS_ISLNK	143
fpS_ISREG	143
fpS_ISSOCK	144
fptime	144
FpTimes	144
FpUmask	145
FpUname	145
FpUnlink	145
FpUtime	146
FpWait	147
FpWaitPid	147
FpWrite	148
wexitStatus	148
wifexited	148
wifsignaled	148
wstopsig	149
wtermSIG	149
2 Reference for unit 'Classes'	150
2.1 Used units	150
2.2 Overview	150
2.3 Constants, types and variables	150
Constants	150
Types	152
Variables	160

2.4	Procedures and functions	161
	BeginGlobalLoading	161
	Bounds	161
	CollectionsEqual	162
	EndGlobalLoading	162
	FindClass	162
	FindNestedComponent	162
	GetClass	162
	GetFixupInstanceNames	163
	GetFixupReferenceNames	163
	GlobalFixupReferences	163
	IdentToInt	164
	InitComponentRes	164
	InitInheritedComponent	164
	IntToIdent	164
	LineStart	165
	NotifyGlobalLoading	165
	ObjectBinaryToText	165
	ObjectResourceToText	165
	ObjectTextToBinary	165
	ObjectTextToResource	166
	Point	166
	ReadComponentRes	166
	ReadComponentResEx	166
	ReadComponentResFile	166
	Rect	167
	RedirectFixupReferences	167
	RegisterClass	167
	RegisterClassAlias	168
	RegisterClasses	168
	RegisterComponents	168
	RegisterInitComponentHandler	168
	RegisterIntegerConsts	169
	RegisterNoIcon	169
	RegisterNonActiveX	169
	RemoveFixupReferences	169
	RemoveFixups	170
	SmallPoint	170
	UnRegisterClass	170
	UnRegisterClasses	170

UnRegisterModuleClasses	171
WriteComponentResFile	171
2.5 EBitsError	171
Description	171
2.6 EClassNotFound	171
Description	171
2.7 EComponentError	171
Description	171
2.8 EFCreateError	172
Description	172
2.9 EFilerError	172
Description	172
2.10 EOpenError	172
Description	172
2.11 EInvalidImage	172
Description	172
2.12 EInvalidOperation	172
Description	172
2.13 EListError	172
Description	172
2.14 EMethodNotFound	173
Description	173
2.15 EOutOfResources	173
Description	173
2.16 EParserError	173
Description	173
2.17 EReadError	173
Description	173
2.18 EResNotFound	173
Description	173
2.19 EStreamError	173
Description	173
2.20 EStringListError	174
Description	174
2.21 EThread	174
Description	174
2.22 EThreadDestroyCalled	174
2.23 EWriteError	174
Description	174
2.24 IStringsAdapter	174

Description	174
2.25 TAbstractObjectReader	174
Description	174
Method overview	175
TAbstractObjectReader.NextValue	175
TAbstractObjectReader.ReadValue	175
TAbstractObjectReader.BeginRootComponent	176
TAbstractObjectReader.BeginComponent	176
TAbstractObjectReader.BeginProperty	176
TAbstractObjectReader.ReadBinary	176
TAbstractObjectReader.ReadFloat	177
TAbstractObjectReader.ReadSingle	177
TAbstractObjectReader.ReadDate	177
TAbstractObjectReader.ReadIdent	178
TAbstractObjectReader.ReadInt8	178
TAbstractObjectReader.ReadInt16	179
TAbstractObjectReader.ReadInt32	179
TAbstractObjectReader.ReadInt64	179
TAbstractObjectReader.ReadSet	180
TAbstractObjectReader.ReadStr	180
TAbstractObjectReader.ReadString	180
TAbstractObjectReader.SkipComponent	181
TAbstractObjectReader.SkipValue	181
2.26 TAbstractObjectWriter	181
Description	181
Method overview	181
TAbstractObjectWriter.BeginCollection	182
TAbstractObjectWriter.BeginComponent	182
TAbstractObjectWriter.BeginList	182
TAbstractObjectWriter.EndList	182
TAbstractObjectWriter.BeginProperty	182
TAbstractObjectWriter.EndProperty	182
TAbstractObjectWriter.WriteBinary	183
TAbstractObjectWriter.WriteBoolean	183
TAbstractObjectWriter.WriteFloat	183
TAbstractObjectWriter.WriteSingle	183
TAbstractObjectWriter.WriteDate	183
TAbstractObjectWriter.WriteIdent	183
TAbstractObjectWriter.WriteInteger	184
TAbstractObjectWriter.WriteMethodName	184

	TAbstractObjectWriter.WriteSet	184
	TAbstractObjectWriter.WriteString	184
2.27	TBasicAction	184
	Description	184
	Method overview	185
	Property overview	185
	TBasicAction.Change	185
	TBasicAction.SetOnExecute	185
	TBasicAction.Create	185
	TBasicAction.Destroy	186
	TBasicAction.HandlesTarget	186
	TBasicAction.UpdateTarget	186
	TBasicAction.ExecuteTarget	187
	TBasicAction.Execute	187
	TBasicAction.RegisterChanges	187
	TBasicAction.UnRegisterChanges	187
	TBasicAction.Update	188
	TBasicAction.OnChange	188
	TBasicAction.ActionComponent	188
	TBasicAction.OnExecute	188
	TBasicAction.OnUpdate	189
2.28	TBasicActionLink	189
	Description	189
	Method overview	189
	Property overview	190
	TBasicActionLink.AssignClient	190
	TBasicActionLink.Change	190
	TBasicActionLink.IsOnExecuteLinked	190
	TBasicActionLink.SetAction	190
	TBasicActionLink.SetOnExecute	191
	TBasicActionLink.Create	191
	TBasicActionLink.Destroy	191
	TBasicActionLink.Execute	191
	TBasicActionLink.Update	192
	TBasicActionLink.Action	192
	TBasicActionLink.OnChange	192
2.29	TBinaryObjectReader	193
	Description	193
	Method overview	193
	TBinaryObjectReader.Create	193

TBinaryObjectReader.Destroy	193
TBinaryObjectReader.NextValue	194
TBinaryObjectReader.ReadValue	194
TBinaryObjectReader.BeginRootComponent	194
TBinaryObjectReader.BeginComponent	194
TBinaryObjectReader.BeginProperty	194
TBinaryObjectReader.ReadBinary	194
TBinaryObjectReader.ReadFloat	194
TBinaryObjectReader.ReadSingle	195
TBinaryObjectReader.ReadDate	195
TBinaryObjectReader.ReadIdent	195
TBinaryObjectReader.ReadInt8	195
TBinaryObjectReader.ReadInt16	195
TBinaryObjectReader.ReadInt32	195
TBinaryObjectReader.ReadInt64	195
TBinaryObjectReader.ReadSet	195
TBinaryObjectReader.ReadStr	195
TBinaryObjectReader.ReadString	196
TBinaryObjectReader.SkipComponent	196
TBinaryObjectReader.SkipValue	196
2.30 TBinaryObjectWriter	196
Description	196
Method overview	196
TBinaryObjectWriter.Create	196
TBinaryObjectWriter.Destroy	197
TBinaryObjectWriter.BeginCollection	197
TBinaryObjectWriter.BeginComponent	197
TBinaryObjectWriter.BeginList	197
TBinaryObjectWriter.EndList	197
TBinaryObjectWriter.BeginProperty	197
TBinaryObjectWriter.EndProperty	198
TBinaryObjectWriter.WriteBinary	198
TBinaryObjectWriter.WriteBoolean	198
TBinaryObjectWriter.WriteFloat	198
TBinaryObjectWriter.WriteSingle	198
TBinaryObjectWriter.WriteDate	198
TBinaryObjectWriter.WriteIdent	198
TBinaryObjectWriter.WriteInteger	199
TBinaryObjectWriter.WriteMethodName	199
TBinaryObjectWriter.WriteSet	199

	TBinaryObjectWriter.WriteString	199
2.31	TBits	199
	Description	199
	Method overview	200
	Property overview	200
	TBits.Create	200
	TBits.Destroy	200
	TBits.GetFSize	201
	TBits.SetOn	201
	TBits.Clear	201
	TBits.Clearall	201
	TBits.AndBits	202
	TBits.OrBits	202
	TBits.XorBits	202
	TBits.NotBits	203
	TBits.Get	203
	TBits.Grow	203
	TBits.Equals	203
	TBits.SetIndex	204
	TBits.FindFirstBit	204
	TBits.FindNextBit	204
	TBits.FindPrevBit	205
	TBits.OpenBit	205
	TBits.Bits	205
	TBits.Size	206
2.32	TCollection	206
	Description	206
	Method overview	206
	Property overview	207
	TCollection.GetAttrCount	207
	TCollection.GetAttr	207
	TCollection.GetItemAttr	207
	TCollection.GetNamePath	207
	TCollection.Changed	208
	TCollection.GetItem	208
	TCollection.SetItem	208
	TCollection.SetItemName	209
	TCollection.SetPropName	209
	TCollection.Update	209
	TCollection.Create	209

TCollection.Destroy	210
TCollection.Add	210
TCollection.Assign	210
TCollection.BeginUpdate	210
TCollection.Clear	211
TCollection.EndUpdate	211
TCollection.FindItemID	211
TCollection.PropName	212
TCollection.Count	212
TCollection.ItemClass	212
TCollection.Items	212
2.33 TCollectionItem	213
Description	213
Method overview	213
Property overview	213
TCollectionItem.Changed	214
TCollectionItem.GetNamePath	214
TCollectionItem.GetOwner	214
TCollectionItem.GetDisplayName	214
TCollectionItem.SetIndex	215
TCollectionItem.SetDisplayName	215
TCollectionItem.Create	215
TCollectionItem.Destroy	215
TCollectionItem.Collection	216
TCollectionItem.ID	216
TCollectionItem.Index	216
TCollectionItem.DisplayName	217
2.34 TComponent	217
Description	217
Method overview	218
Property overview	219
TComponent.ChangeName	219
TComponent.DefineProperties	219
TComponent.GetChildren	219
TComponent.GetChildOwner	220
TComponent.GetChildParent	220
TComponent.GetNamePath	220
TComponent.GetOwner	220
TComponent.Loaded	221
TComponent.Notification	221

TComponent.ReadState	221
TComponent.SetAncestor	222
TComponent.SetDesigning	222
TComponent.SetName	222
TComponent.SetChildOrder	223
TComponent.SetParentComponent	223
TComponent.Updating	223
TComponent.Updated	223
TComponent.UpdateRegistry	224
TComponent.ValidateRename	224
TComponent.ValidateContainer	224
TComponent.ValidateInsert	224
TComponent.WriteState	225
TComponent.Create	225
TComponent.BeforeDestruction	225
TComponent.Destroy	225
TComponent.DestroyComponents	226
TComponent.Destroying	226
TComponent.ExecuteAction	226
TComponent.FindComponent	226
TComponent.FreeNotification	227
TComponent.RemoveFreeNotification	227
TComponent.FreeOnRelease	227
TComponent.GetParentComponent	227
TComponent.HasParent	227
TComponent.InsertComponent	228
TComponent.RemoveComponent	228
TComponent.SafeCallException	228
TComponent.UpdateAction	228
TComponent.Components	228
TComponent.ComponentCount	229
TComponent.ComponentIndex	229
TComponent.ComponentState	229
TComponent.ComponentStyle	230
TComponent.DesignInfo	230
TComponent.Owner	230
TComponent.VCLComObject	230
TComponent.Name	231
TComponent.Tag	231
2.35 TCustomMemoryStream	231

Description	231
Method overview	232
Property overview	232
TCustomMemoryStream.SetPointer	232
TCustomMemoryStream.Read	232
TCustomMemoryStream.Seek	232
TCustomMemoryStream.SaveToStream	233
TCustomMemoryStream.SaveToFile	233
TCustomMemoryStream.Memory	234
2.36 TDataModule	234
Method overview	234
Property overview	234
TDataModule.DoCreate	234
TDataModule.DoDestroy	234
TDataModule.DefineProperties	235
TDataModule.GetChildren	235
TDataModule.HandleCreateException	235
TDataModule.ReadState	235
TDataModule.Create	235
TDataModule.CreateNew	235
TDataModule.Destroy	235
TDataModule.AfterConstruction	235
TDataModule.BeforeDestruction	235
TDataModule.DesignOffset	236
TDataModule.DesignSize	236
TDataModule.OnCreate	236
TDataModule.OnDestroy	236
TDataModule.OldCreateOrder	236
2.37 TFiler	236
Description	236
Method overview	236
Property overview	237
TFiler.SetRoot	237
TFiler.DefineProperty	237
TFiler.DefineBinaryProperty	237
TFiler.Root	237
TFiler.LookupRoot	238
TFiler.Ancestor	238
TFiler.IgnoreChildren	238
2.38 TFileStream	238

Description	238
Method overview	238
Property overview	239
TFileStream.Create	239
TFileStream.Destroy	239
TFileStream.FileName	239
2.39 THandleStream	240
Description	240
Method overview	240
Property overview	240
THandleStream.SetSize	240
THandleStream.Create	240
THandleStream.Read	241
THandleStream.Write	241
THandleStream.Seek	241
THandleStream.Handle	241
2.40 TList	242
Description	242
Method overview	242
Property overview	242
TList.Get	242
TList.Grow	243
TList.Put	243
TList.Notify	243
TList.SetCapacity	243
TList.SetCount	243
TList.RaiseIndexError	243
TList.Destroy	243
TList.Add	244
TList.Clear	244
TList.Delete	244
TList.Error	244
TList.Exchange	244
TList.Expand	245
TList.Extract	245
TList.First	245
TList.IndexOf	245
TList.Insert	246
TList.Last	246
TList.Move	246

TList.Assign	246
TList.Remove	246
TList.Pack	247
TList.Sort	247
TList.Capacity	247
TList.Count	248
TList.Items	248
TList.List	248
2.41 TMemoryStream	248
Description	248
Method overview	249
Property overview	249
TMemoryStream.Realloc	249
TMemoryStream.Destroy	249
TMemoryStream.Clear	249
TMemoryStream.LoadFromStream	250
TMemoryStream.LoadFromFile	250
TMemoryStream.SetSize	250
TMemoryStream.Write	251
TMemoryStream.Capacity	251
2.42 TOwnedCollection	251
Method overview	251
TOwnedCollection.GetOwner	251
TOwnedCollection.Create	251
2.43 TParser	252
Description	252
Method overview	252
Property overview	252
TParser.Create	252
TParser.Destroy	252
TParser.CheckToken	253
TParser.CheckTokenSymbol	253
TParser.Error	253
TParser.ErrorFmt	253
TParser.ErrorStr	253
TParser.HexToBinary	253
TParser.NextToken	254
TParser.SourcePos	254
TParser.TokenComponentIdent	254
TParser.TokenFloat	254

TParser.TokenInt	254
TParser.TokenString	254
TParser.TokenSymbols	255
TParser.SourceLine	255
TParser.Token	255
2.44 TPersistent	255
Description	255
Method overview	255
TPersistent.AssignTo	256
TPersistent.DefineProperties	256
TPersistent.GetOwner	257
TPersistent.Destroy	257
TPersistent.Assign	257
TPersistent.GetNamePath	258
2.45 TReader	258
Description	258
Method overview	259
Property overview	260
TReader.Error	260
TReader.FindMethod	260
TReader.ReadProperty	261
TReader.ReadPropValue	261
TReader.PropertyError	261
TReader.ReadData	261
TReader.Create	261
TReader.Destroy	261
TReader.BeginReferences	262
TReader.CheckValue	262
TReader.DefineProperty	262
TReader.DefineBinaryProperty	262
TReader.EndOfList	262
TReader.EndReferences	263
TReader.FixupReferences	263
TReader.NextValue	263
TReader.ReadBoolean	263
TReader.ReadChar	263
TReader.ReadCollection	263
TReader.ReadComponent	264
TReader.ReadComponents	264
TReader.ReadFloat	264

TReader.ReadSingle	264
TReader.ReadDate	264
TReader.ReadIdent	264
TReader.ReadInteger	265
TReader.ReadInt64	265
TReader.ReadListBegin	265
TReader.ReadListEnd	265
TReader.ReadRootComponent	265
TReader.ReadString	265
TReader.ReadValue	266
TReader.CopyValue	266
TReader.PropName	266
TReader.CanHandleExceptions	266
TReader.Driver	266
TReader.Owner	267
TReader.Parent	267
TReader.OnError	267
TReader.OnPropertyNotFound	267
TReader.OnFindMethod	267
TReader.OnSetMethodProperty	268
TReader.OnSetName	268
TReader.OnReferenceName	268
TReader.OnAncestorNotFound	268
TReader.OnCreateComponent	268
TReader.OnFindComponentClass	269
TReader.OnReadStringProperty	269
2.46 TRecall	269
Method overview	269
Property overview	269
TRecall.Create	269
TRecall.Destroy	269
TRecall.Store	269
TRecall.Forget	270
TRecall.Reference	270
2.47 TResourceStream	270
Description	270
Method overview	270
TResourceStream.Create	270
TResourceStream.CreateFromID	270
TResourceStream.Destroy	271

	TResourceStream.Write	271
2.48	TStream	271
	Description	271
	Method overview	272
	Property overview	272
	TStream.SetSize	272
	TStream.Read	272
	TStream.Write	273
	TStream.Seek	273
	TStream.ReadBuffer	274
	TStream.WriteBuffer	274
	TStream.CopyFrom	274
	TStream.ReadComponent	275
	TStream.ReadComponentRes	275
	TStream.WriteComponent	275
	TStream.WriteComponentRes	276
	TStream.WriteDescendent	276
	TStream.WriteDescendentRes	276
	TStream.WriteResourceHeader	276
	TStream.FixupResourceHeader	277
	TStream.ReadResHeader	277
	TStream.ReadByte	277
	TStream.ReadWord	277
	TStream.ReadDWord	278
	TStream.ReadAnsiString	278
	TStream.WriteByte	278
	TStream.WriteWord	279
	TStream.WriteDWord	279
	TStream.WriteAnsiString	279
	TStream.Position	280
	TStream.Size	280
2.49	TStringList	280
	Description	280
	Method overview	281
	Property overview	281
	TStringList.Changed	281
	TStringList.Changing	281
	TStringList.Get	282
	TStringList.GetCapacity	282
	TStringList.GetCount	282

TStringList.GetObject	282
TStringList.Put	282
TStringList.PutObject	283
TStringList.SetCapacity	283
TStringList.SetUpdateState	283
TStringList.Destroy	283
TStringList.Add	283
TStringList.Clear	284
TStringList.Delete	284
TStringList.Exchange	284
TStringList.Find	284
TStringList.IndexOf	285
TStringList.Insert	285
TStringList.Sort	285
TStringList.CustomSort	285
TStringList.Duplicates	285
TStringList.Sorted	286
TStringList.OnChange	286
TStringList.OnChanging	286
2.50 TStringList	287
Description	287
Method overview	288
Property overview	288
TStringList.DefineProperties	289
TStringList.Error	289
TStringList.Get	289
TStringList.GetCapacity	289
TStringList.GetCount	290
TStringList.GetObject	290
TStringList.GetTextStr	290
TStringList.Put	290
TStringList.PutObject	291
TStringList.SetCapacity	291
TStringList.SetTextStr	291
TStringList.SetUpdateState	292
TStringList.Destroy	292
TStringList.Add	292
TStringList.AddObject	292
TStringList.Append	293
TStringList.AddStrings	293

TStrings.Assign	293
TStrings.BeginUpdate	293
TStrings.Clear	294
TStrings.Delete	294
TStrings.EndUpdate	294
TStrings.Equals	295
TStrings.Exchange	295
TStrings.GetText	295
TStrings.IndexOf	295
TStrings.IndexOfName	296
TStrings.IndexOfObject	296
TStrings.Insert	296
TStrings.InsertObject	297
TStrings.LoadFromFile	297
TStrings.LoadFromStream	297
TStrings.Move	298
TStrings.SaveToFile	298
TStrings.SaveToStream	299
TStrings.SetText	299
TStrings.Capacity	299
TStrings.CommaText	299
TStrings.Count	300
TStrings.Names	300
TStrings.Objects	301
TStrings.Values	301
TStrings.Strings	302
TStrings.Text	302
TStrings.StringsAdapter	302
2.51 TStringStream	303
Description	303
Method overview	303
Property overview	303
TStringStream.SetSize	303
TStringStream.Create	303
TStringStream.Read	304
TStringStream.ReadString	304
TStringStream.Seek	304
TStringStream.Write	304
TStringStream.WriteString	304
TStringStream.DataString	305

2.52	TTextObjectWriter	305
	Description	305
2.53	TThread	305
	Description	305
	Method overview	305
	Property overview	305
	TThread.DoTerminate	306
	TThread.Execute	306
	TThread.Synchronize	306
	TThread.Create	306
	TThread.Destroy	306
	TThread.Resume	306
	TThread.Suspend	307
	TThread.Terminate	307
	TThread.WaitFor	307
	TThread.ReturnValue	307
	TThread.Terminated	307
	TThread.FreeOnTerminate	308
	TThread.Handle	308
	TThread.Priority	308
	TThread.Suspended	308
	TThread.ThreadID	308
	TThread.OnTerminate	309
	TThread.FatalException	309
2.54	TThreadList	309
	Description	309
	Method overview	309
	TThreadList.Create	309
	TThreadList.Destroy	309
	TThreadList.Add	310
	TThreadList.Clear	310
	TThreadList.LockList	310
	TThreadList.Remove	310
	TThreadList.UnlockList	311
2.55	TWriter	311
	Description	311
	Method overview	311
	Property overview	311
	TWriter.SetRoot	312
	TWriter.WriteBinary	312

TWriter.WriteProperty	312
TWriter.WriteProperties	312
TWriter.Create	312
TWriter.Destroy	312
TWriter.DefineProperty	313
TWriter.DefineBinaryProperty	313
TWriter.WriteBoolean	313
TWriter.WriteCollection	313
TWriter.WriteComponent	313
TWriter.WriteChar	314
TWriter.WriteDescendent	314
TWriter.WriteFloat	314
TWriter.WriteSingle	314
TWriter.WriteDate	314
TWriter.WriteIdent	314
TWriter.WriteInteger	315
TWriter.WriteListBegin	315
TWriter.WriteListEnd	315
TWriter.WriteRootComponent	315
TWriter.WriteString	315
TWriter.RootAncestor	316
TWriter.OnFindAncestor	316
TWriter.OnWriteMethodProperty	316
TWriter.OnWriteStringProperty	316
TWriter.Driver	316
3 Reference for unit 'Crt'	317
3.1 Overview	317
3.2 Constants, types and variables	317
Constants	317
Types	320
Variables	320
3.3 Procedures and functions	321
AssignCrt	321
ClrEol	321
ClrScr	322
cursorbig	323
cursoroff	323
cursoron	323
Delay	323

DelLine	324
GotoXY	325
HighVideo	325
InsLine	326
KeyPressed	326
LowVideo	327
NormVideo	327
NoSound	327
ReadKey	328
Sound	329
TextBackground	329
TextColor	330
TextMode	330
WhereX	330
WhereY	331
Window	331
4 Reference for unit 'dateutils'	333
4.1 Used units	333
4.2 Overview	333
4.3 Constants, types and variables	333
Constants	333
4.4 Procedures and functions	335
CompareDate	335
CompareDateTime	336
CompareTime	337
DateOf	338
DateTimeToJulianDate	339
DateTimeToModifiedJulianDate	339
DateTimeToUnix	339
DayOf	339
DayOfTheMonth	340
DayOfTheWeek	340
DayOfTheYear	341
DaysBetween	341
DaysInAMonth	342
DaysInAYear	342
DaysInMonth	343
DaysInYear	343
DaySpan	344

DecodeDateDay	345
DecodeDateMonthWeek	345
DecodeDateTime	346
DecodeDateWeek	346
DecodeDayOfWeekInMonth	347
EncodeDateDay	348
EncodeDateMonthWeek	348
EncodeDateTime	348
EncodeDateWeek	349
EncodeDayOfWeekInMonth	349
EndOfADay	350
EndOfAMonth	350
EndOfAWeek	351
EndOfAYear	352
EndOfTheDay	352
EndOfTheMonth	353
EndOfTheWeek	353
EndOfTheYear	354
HourOf	354
HourOfTheDay	354
HourOfTheMonth	355
HourOfTheWeek	355
HourOfTheYear	356
HoursBetween	356
HourSpan	357
IncDay	358
IncHour	358
IncMilliSecond	359
IncMinute	359
IncSecond	360
IncWeek	360
IncYear	361
InvalidDateDayError	361
InvalidDateMonthWeekError	361
InvalidDateTimeError	362
InvalidDateWeekError	362
InvalidDayOfWeekInMonthError	363
IsInLeapYear	363
IsPM	363
IsSameDay	364

IsToday	364
IsValidDate	365
IsValidDateDay	365
IsValidDateMonthWeek	366
IsValidDateTime	367
IsValidDateWeek	368
IsValidTime	369
JulianDateToDateTime	369
MilliSecondOf	369
MilliSecondOfTheDay	369
MilliSecondOfTheHour	370
MilliSecondOfTheMinute	370
MilliSecondOfTheMonth	370
MilliSecondOfTheSecond	370
MilliSecondOfTheWeek	371
MilliSecondOfTheYear	371
MilliSecondsBetween	372
MilliSecondSpan	372
MinuteOf	373
MinuteOfTheDay	373
MinuteOfTheHour	374
MinuteOfTheMonth	374
MinuteOfTheWeek	375
MinuteOfTheYear	375
MinutesBetween	375
MinuteSpan	376
ModifiedJulianDateToDateTime	377
MonthOf	377
MonthOfTheYear	377
MonthsBetween	378
MonthSpan	379
NthDayOfWeek	379
PreviousDayOfWeek	380
RecodeDate	380
RecodeDateTime	381
RecodeDay	382
RecodeHour	383
RecodeMilliSecond	383
RecodeMinute	384
RecodeMonth	385

RecodeSecond	385
RecodeTime	386
RecodeYear	387
SameDate	387
SameDateTime	388
SameTime	389
SecondOf	390
SecondOfTheDay	390
SecondOfTheHour	390
SecondOfTheMinute	391
SecondOfTheMonth	391
SecondOfTheWeek	391
SecondOfTheYear	392
SecondsBetween	392
SecondSpan	393
StartOfADay	394
StartOfAMonth	395
StartOfAWeek	395
StartOfAYear	396
StartOfTheDay	396
StartOfTheMonth	397
StartOfTheWeek	397
StartOfTheYear	398
TimeOf	398
Today	399
Tomorrow	399
TryEncodeDateDay	400
TryEncodeDateMonthWeek	400
TryEncodeDateTime	401
TryEncodeDateWeek	402
TryEncodeDayOfWeekInMonth	402
TryJulianDateToDateTime	403
TryModifiedJulianDateToDateTime	403
TryRecodeDateTime	404
UnixToDateTime	405
WeekOf	405
WeekOfTheMonth	405
WeekOfTheYear	406
WeeksBetween	407
WeeksInAYear	407

WeeksInYear	408
WeekSpan	409
WithinPastDays	409
WithinPastHours	410
WithinPastMilliSeconds	411
WithinPastMinutes	412
WithinPastMonths	413
WithinPastSeconds	414
WithinPastWeeks	415
WithinPastYears	416
YearOf	417
YearsBetween	418
YearSpan	419
Yesterday	420
5 Reference for unit 'Dos'	421
5.1 System information	421
5.2 Process handling	421
5.3 Directory and disk handling	421
5.4 File handling	422
5.5 File open mode constants	422
5.6 File attributes	422
5.7 Overview	423
5.8 Constants, types and variables	423
Constants	423
Types	425
Variables	427
5.9 Procedures and functions	427
AddDisk	427
DiskFree	428
DiskSize	429
DosExitCode	429
DosVersion	430
DTToUnixDate	430
EnvCount	431
EnvStr	431
Exec	432
FExpand	432
FindClose	432
FindFirst	433

FindNext	434
FSearch	434
FSplit	434
GetCBreak	435
GetDate	435
GetEnv	436
GetFAttr	436
GetFTime	437
GetIntVec	438
GetLongName	438
GetMsCount	439
GetShortName	439
GetTime	439
GetVerify	440
Intr	440
Keep	441
MSDos	441
PackTime	441
SetCBreak	442
SetDate	442
SetFAttr	443
SetFTime	443
SetIntVec	443
SetTime	444
SetVerify	444
SwapVectors	444
UnixDateToDt	445
UnpackTime	445
weekday	445
6 Reference for unit 'dxeload'	446
6.1 Overview	446
6.2 Procedures and functions	446
dxeload	446
7 Reference for unit 'dynlibs'	447
7.1 Overview	447
7.2 Constants, types and variables	447
Constants	447
Types	447
7.3 Procedures and functions	447

FreeLibrary	447
GetProcAddress	448
GetProcedureAddress	448
LoadLibrary	448
UnloadLibrary	448
8 Reference for unit 'emu387'	450
8.1 Overview	450
8.2 Procedures and functions	450
npxsetup	450
9 Reference for unit 'getopts'	451
9.1 Overview	451
9.2 Constants, types and variables	451
Constants	451
Types	451
Variables	452
9.3 Procedures and functions	452
GetLongOpts	452
GetOpt	453
10 Reference for unit 'go32'	456
10.1 Real mode callbacks	456
10.2 Executing software interrupts	457
10.3 Software interrupts	457
10.4 Hardware interrupts	457
10.5 Disabling interrupts	457
10.6 Creating your own interrupt handlers	457
10.7 Protected mode interrupts vs. Real mode interrupts	458
10.8 Handling interrupts with DPMI	458
10.9 Interrupt redirection	458
10.10 Processor access	458
10.11 I/O port access	458
10.12 dos memory access	458
10.13 FPC specialities	459
10.14 Selectors and descriptors	459
10.15 What is DPMI	459
10.16 Overview	459
10.17 Constants, types and variables	460
Constants	460
Types	462

Variables	463
10.18 Procedures and functions	463
allocate_ldt_descriptors	463
allocate_memory_block	465
copyfromdos	466
copytodos	466
create_code_segment_alias_descriptor	467
disable	467
dpmi_dosmemfillchar	467
dpmi_dosmemfillword	467
dpmi_dosmemget	467
dpmi_dosmemmove	468
dpmi_dosmemput	468
enable	468
free_ldt_descriptor	468
free_memory_block	469
free_rm_callback	469
get_cs	469
get_descriptor_access_right	470
get_ds	470
get_exception_handler	470
get_linear_addr	470
get_meminfo	471
get_next_selector_increment_value	472
get_page_size	472
get_pm_exception_handler	472
get_pm_interrupt	473
get_rm_callback	473
get_rm_interrupt	476
get_run_mode	476
get_segment_base_address	477
get_segment_limit	478
get_ss	478
global_dos_alloc	478
global_dos_free	480
inportb	480
inportl	481
inportw	481
lock_code	481
lock_data	482

lock_linear_region	482
map_device_in_memory_block	482
outportb	483
outportl	483
outportw	484
realintr	484
request_linear_region	485
segment_to_descriptor	485
seg_fillchar	485
seg_fillword	486
seg_move	486
set_descriptor_access_right	487
set_exception_handler	487
set_pm_exception_handler	487
set_pm_interrupt	487
set_rm_interrupt	488
set_segment_base_address	489
set_segment_limit	489
tb_offset	490
tb_segment	490
tb_size	490
transfer_buffer	490
unlock_code	490
unlock_data	491
unlock_linear_region	491
11 Reference for unit 'gpm'	492
11.1 Used units	492
11.2 Overview	492
11.3 Constants, types and variables	492
Constants	492
Types	494
Variables	496
11.4 Procedures and functions	497
Gpm_AnyDouble	497
Gpm_AnySingle	497
Gpm_AnyTriple	497
gpm_close	497
gpm_fitvalues	498
gpm_fitvaluesM	498

gpm_getevent	498
gpm_getsnapshot	499
gpm_lowerroi	500
gpm_open	500
gpm_poproi	500
gpm_pushroi	501
gpm_raiseroi	501
gpm_repeat	501
Gpm_StrictDouble	501
Gpm_StrictSingle	502
Gpm_StrictTriple	502
12 Reference for unit 'Graph'	503
12.1 Categorized functions: Text and font handling	503
12.2 Categorized functions: Filled drawings	503
12.3 Categorized functions: Drawing primitives	503
12.4 Categorized functions: Color management	503
12.5 Categorized functions: Screen management	504
12.6 Categorized functions: Initialization	504
12.7 Target specific issues: Linux	504
12.8 Target specific issues: DOS	506
12.9 A word about mode selection	506
12.10Requirements	508
12.11Overview	508
12.12Constants, types and variables	508
Constants	508
Types	522
Variables	527
12.13Procedures and functions	529
Arc	529
Bar	529
Bar3D	530
ClearDevice	530
Closegraph	530
DetectGraph	530
DrawPoly	531
Ellipse	531
FillEllipse	531
FillPoly	531
FloodFill	532

GetArcCoords	532
GetAspectRatio	532
GetBkColor	532
GetColor	533
GetDefaultPalette	533
GetDirectVideo	533
GetDriverName	533
GetFillPattern	534
GetFillSettings	534
GetGraphMode	534
GetLineSettings	534
GetMaxColor	535
GetMaxMode	535
GetMaxX	535
GetMaxY	535
GetModeName	536
GetModeRange	536
GetPalette	536
GetPaletteSize	536
GetTextSettings	537
GetViewSettings	537
GetX	537
GetY	537
GraphDefaults	538
GraphErrorMsg	538
GraphResult	538
InitGraph	539
InstallUserDriver	539
InstallUserFont	540
LineRel	540
LineTo	540
MoveRel	540
MoveTo	541
OutText	541
PieSlice	541
queryadapterinfo	541
Rectangle	542
RegisterBGIDriver	542
RegisterBGIfont	542
RestoreCrtMode	542

Sector	543
SetAspectRatio	543
SetBkColor	543
SetColor	543
SetDirectVideo	544
SetFillPattern	544
SetFillStyle	544
SetGraphMode	545
SetLineStyle	545
SetPalette	545
SetTextJustify	546
SetTextStyle	546
SetUserCharSize	547
SetViewPort	547
SetWriteMode	547
TextHeight	548
TextWidth	548
13 Reference for unit 'heaptrc'	549
13.1 Controlling HeapTrc with environment variables	549
13.2 HeapTrc Usage	549
13.3 Overview	550
13.4 Constants, types and variables	550
Constants	550
Types	551
13.5 Procedures and functions	551
DumpHeap	551
MarkHeap	552
SetHeapExtraInfo	552
SetHeapTraceOutput	553
14 Reference for unit 'ipc'	555
14.1 Used units	555
14.2 Overview	555
14.3 Constants, types and variables	555
Constants	555
Types	558
14.4 Procedures and functions	561
ftok	561
msgctl	562
msgget	564

msgrcv	565
msgsnd	565
semctl	566
semget	570
semop	571
shmat	571
shmctl	572
shmdt	574
shmget	574
15 Reference for unit 'Keyboard'	576
15.1 Writing a keyboard driver	576
15.2 Keyboard scan codes	577
15.3 Overview	577
15.4 Constants, types and variables	577
Constants	577
Types	582
15.5 Procedures and functions	582
AddSequence	582
DoneKeyboard	582
FindSequence	583
FunctionKeyName	583
GetKeyboardDriver	584
GetKeyEvent	584
GetKeyEventChar	585
GetKeyEventCode	585
GetKeyEventFlags	586
GetKeyEventShiftState	586
GetKeyEventUniCode	587
InitKeyboard	587
IsFunctionKey	588
KeyEventToString	589
KeyPressed	589
PollKeyEvent	589
PollShiftStateEvent	590
PutKeyEvent	591
RawReadKey	592
RawReadString	592
RestoreStartMode	592
SetKeyboardDriver	592

ShiftStateToString	592
TranslateKeyEvent	593
TranslateKeyEventUniCode	593
16 Reference for unit 'Linux'	596
16.1 Overview	596
16.2 Constants, types and variables	596
Constants	596
Types	597
16.3 Procedures and functions	597
Clone	597
Sysinfo	599
17 Reference for unit 'math'	601
17.1 Geometrical functions	601
17.2 Statistical functions	601
17.3 Number converting	602
17.4 Exponential and logarithmic functions	602
17.5 Hyperbolic functions	602
17.6 Trigonometric functions	602
17.7 Angle unit conversion	602
17.8 Min/max determination	603
17.9 Used units	603
17.10 Overview	603
17.11 Constants, types and variables	604
Constants	604
Types	605
17.12 Procedures and functions	606
arccos	606
arccosh	607
arcosh	607
arcsin	608
arcsinh	608
arctan2	609
arctanh	609
arsinh	610
artanh	610
ceil	611
ClearExceptions	611
cosh	611
cotan	612

cycletorad	612
degtograd	613
degtorad	613
DivMod	614
EnsureRange	614
floor	614
Frexp	615
GetExceptionMask	616
GetPrecisionMode	616
GetRoundMode	616
GetSSECSR	616
gradtodeg	616
gradtorad	617
hypot	617
InRange	618
intpower	618
IsInfinite	619
IsNan	619
IsZero	619
ldexp	620
lnxp1	620
log10	621
log2	621
logn	622
Max	622
MaxIntValue	623
maxvalue	624
mean	624
meanandstddev	625
Min	626
MinIntValue	627
minvalue	627
momentskewkurtosis	628
norm	629
popnstddev	630
popnvariance	630
power	631
radtoycle	632
radtodeg	632
radtoegrad	633

randg	633
SameValue	634
SetExceptionMask	634
SetPrecisionMode	635
SetRoundMode	635
SetSSECSR	635
Sign	635
sincos	635
sinh	636
stddev	637
sum	637
sumofsquares	638
sumsandsquares	639
tan	640
tanh	640
totalvariance	641
variance	641
17.13invalidargument	642
Description	642
18 Reference for unit 'mmx'	643
18.1 Overview	643
18.2 Constants, types and variables	643
Constants	643
Types	644
18.3 Procedures and functions	645
emms	645
femms	645
19 Reference for unit 'Mouse'	646
19.1 Writing a custom mouse driver	646
19.2 Overview	646
19.3 Constants, types and variables	646
Constants	646
Types	647
Variables	648
19.4 Procedures and functions	648
DetectMouse	648
DoneMouse	649
GetMouseButtons	649
GetMouseDriver	650

MouseEvent	650
GetX	650
GetY	651
HideMouse	651
InitMouse	652
PollMouseEvent	653
PutMouseEvent	653
SetMouseDriver	653
SetMouseXY	654
ShowMouse	654
20 Reference for unit 'Objects'	655
20.1 Overview	655
20.2 Constants, types and variables	655
Constants	655
Types	657
Variables	660
20.3 Procedures and functions	661
Abstract	661
CallPointerConstructor	661
CallPointerLocal	661
CallPointerMethod	661
CallPointerMethodLocal	661
CallVoidConstructor	661
CallVoidLocal	662
CallVoidMethod	662
CallVoidMethodLocal	662
DisposeStr	662
LongDiv	662
LongMul	662
NewStr	663
RegisterObjects	663
RegisterType	664
SetStr	665
20.4 TBufStream	666
Description	666
Method overview	666
TBufStream.Init	666
TBufStream.Done	667
TBufStream.Close	667

TBufStream.Flush	667
TBufStream.Truncate	668
TBufStream.Seek	668
TBufStream.Open	669
TBufStream.Read	669
TBufStream.Write	669
20.5 TCollection	670
Description	670
Method overview	670
TCollection.Init	670
TCollection.Load	671
TCollection.Done	672
TCollection.At	672
TCollection.IndexOf	673
TCollection.GetItem	673
TCollection.LastThat	674
TCollection.FirstThat	675
TCollection.Pack	675
TCollection.FreeAll	676
TCollection.DeleteAll	677
TCollection.Free	678
TCollection.Insert	679
TCollection.Delete	679
TCollection.AtFree	680
TCollection.FreeItem	681
TCollection.AtDelete	681
TCollection.ForEach	682
TCollection.SetLimit	683
TCollection.Error	683
TCollection.AtPut	683
TCollection.AtInsert	684
TCollection.Store	684
TCollection.PutItem	685
20.6 TDosStream	685
Description	685
Method overview	685
TDosStream.Init	686
TDosStream.Done	686
TDosStream.Close	686
TDosStream.Truncate	687

TDosStream.Seek	687
TDosStream.Open	688
TDosStream.Read	689
TDosStream.Write	689
20.7 TMemoryStream	690
Description	690
Method overview	690
TMemoryStream.Init	690
TMemoryStream.Done	690
TMemoryStream.Truncate	691
TMemoryStream.Read	691
TMemoryStream.Write	692
20.8 TObject	692
Description	692
Method overview	692
TObject.Init	692
TObject.Free	692
TObject.Is_Object	693
TObject.Done	693
20.9 TPoint	694
Description	694
20.10 TRect	694
Description	694
Method overview	694
TRect.Empty	694
TRect.Equals	695
TRect.Contains	696
TRect.Copy	696
TRect.Union	697
TRect.Intersect	697
TRect.Move	698
TRect.Grow	699
TRect.Assign	699
20.11 TResourceCollection	700
Description	700
Method overview	700
TResourceCollection.KeyOf	700
TResourceCollection.GetItem	701
TResourceCollection.FreeItem	701
TResourceCollection.PutItem	701

20.12	TResourceFile	701
	Description	701
	Method overview	702
	TResourceFile.Init	702
	TResourceFile.Done	702
	TResourceFile.Count	702
	TResourceFile.KeyAt	703
	TResourceFile.Get	703
	TResourceFile.SwitchTo	703
	TResourceFile.Flush	703
	TResourceFile.Delete	704
	TResourceFile.Put	704
20.13	TSortedCollection	704
	Description	704
	Method overview	705
	TSortedCollection.Init	705
	TSortedCollection.Load	705
	TSortedCollection.KeyOf	705
	TSortedCollection.IndexOf	706
	TSortedCollection.Compare	706
	TSortedCollection.Search	707
	TSortedCollection.Insert	708
	TSortedCollection.Store	709
20.14	TStrCollection	710
	Description	710
	Method overview	710
	TStrCollection.Compare	710
	TStrCollection.GetItem	711
	TStrCollection.FreeItem	711
	TStrCollection.PutItem	711
20.15	TStream	712
	Description	712
	Method overview	712
	TStream.Init	712
	TStream.Get	712
	TStream.StrRead	713
	TStream.GetPos	714
	TStream.GetSize	714
	TStream.ReadStr	715
	TStream.Open	716

TStream.Close	716
TStream.Reset	716
TStream.Flush	717
TStream.Truncate	717
TStream.Put	717
TStream.StrWrite	718
TStream.WriteStr	718
TStream.Seek	718
TStream.Error	718
TStream.Read	719
TStream.Write	719
TStream.CopyFrom	720
20.16 TStringCollection	720
Description	720
Method overview	721
TStringCollection.GetItem	721
TStringCollection.Compare	721
TStringCollection.FreeItem	722
TStringCollection.PutItem	722
20.17 TStringList	722
Description	722
Method overview	723
TStringList.Load	723
TStringList.Done	723
TStringList.Get	723
20.18 TStrListMaker	724
Description	724
Method overview	724
TStrListMaker.Init	724
TStrListMaker.Done	724
TStrListMaker.Put	724
TStrListMaker.Store	725
20.19 TUnSortedStrCollection	725
Description	725
Method overview	725
TUnSortedStrCollection.Insert	725
21 Reference for unit 'objpas'	727
21.1 Overview	727
21.2 Constants, types and variables	727

Constants	727
Types	727
21.3 Procedures and functions	728
AssignFile	728
CloseFile	729
GetResourceStringCurrentValue	729
GetResourceStringDefaultValue	730
GetResourceStringHash	731
GetResourceStringName	731
Hash	732
LoadResString	733
ParamStr	733
ResetResourceTables	733
ResourceStringCount	734
ResourceStringTableCount	734
SetResourceStrings	734
SetResourceStringValue	735
22 Reference for unit 'oldlinux'	737
22.1 Utility routines	737
22.2 Terminal functions	737
22.3 System information	737
22.4 Signals	738
22.5 Process handling	738
22.6 Directory handling routines	738
22.7 Pipes, FIFOs and streams	739
22.8 General File handling routines	739
22.9 File Input/Output routines	739
22.10 Overview	739
22.11 Constants, types and variables	739
Constants	739
Types	778
Variables	787
22.12 Procedures and functions	787
Access	787
Alarm	788
AssignPipe	789
AssignStream	790
Basename	791
CFMakeRaw	792

CFSetISpeed	792
CFSetOSpeed	792
Chmod	792
Chown	794
Clone	795
CloseDir	797
CreateShellArgV	797
Dirname	798
Dup	798
Dup2	799
EpochToLocal	800
Execl	800
Execle	801
Execlp	802
Execv	803
Execve	804
Execvp	805
ExitProcess	806
Fcntl	806
fdClose	807
fdFlush	807
fdOpen	807
fdRead	808
fdSeek	809
fdTruncate	810
fdWrite	810
FD_Clr	810
FD_IsSet	811
FD_Set	811
FD_Zero	811
FExpand	811
Flock	812
FNMatch	812
Fork	813
FReName	813
FSearch	814
FSplit	815
FSSStat	815
FStat	816
GetDate	817

GetDateTime	818
GetDomainName	818
GetEGid	819
GetEnv	819
GetEpochTime	820
GetEUid	820
GetFS	821
GetGid	821
GetHostName	822
GetLocalTimezone	822
GetPid	823
GetPPid	823
GetPriority	824
GetTime	824
GetTimeOfDay	825
GetTimezoneFile	825
GetUid	825
Glob	826
Globfree	827
IOCtl	827
IOperm	827
IoPL	828
IsATTY	828
Kill	828
Link	829
LocalToEpoch	830
Lstat	831
mkFifo	832
MMap	832
MUnMap	834
NanoSleep	834
Nice	835
Octal	836
OpenDir	836
Pause	837
PClose	838
POpen	838
ReadDir	839
ReadLink	839
ReadTimezoneFile	840

SeekDir	841
Select	841
SelectText	842
SetDate	842
SetDateTime	843
SetPriority	843
SetTime	844
Shell	844
SigAction	844
Signal	846
SigPending	847
SigProcMask	847
SigRaise	847
SigSuspend	848
StringToPPChar	848
SymLink	849
SysCall	850
Sysinfo	851
S_ISBLK	852
S_ISCHR	852
S_ISDIR	852
S_ISFIFO	853
S_ISLNK	853
S_ISREG	854
S_ISSOCK	854
TCDrain	854
TCFlow	854
TCFlush	855
TCGetAttr	855
TCGetPGrp	856
TCSendBreak	856
TCSetAttr	857
TCSetPGrp	857
TellDir	857
TTYname	858
Umask	858
Uname	858
UnLink	859
Utime	859
WaitPid	860

WaitProcess	861
WEXITSTATUS	861
WIFEXITED	861
WIFSIGNALED	861
WIFSTOPPED	862
WSTOPSIG	862
WTERMSIG	862
W_EXITCODE	862
W_STOPCODE	863
23 Reference for unit 'ports'	864
23.1 Overview	864
23.2 Constants, types and variables	864
Variables	864
23.3 tport	865
Description	865
Method overview	865
Property overview	865
tport.writeport	865
tport.readport	866
tport.pp	866
23.4 tportl	866
Description	866
Method overview	866
Property overview	866
tportl.writeport	866
tportl.readport	867
tportl.pp	867
23.5 tportw	867
Description	867
Method overview	867
Property overview	867
tportw.writeport	867
tportw.readport	868
tportw.pp	868
24 Reference for unit 'printer'	869
24.1 Overview	869
24.2 Constants, types and variables	869
Variables	869
24.3 Procedures and functions	869

AssignLst	869
InitPrinter	870
IsLstAvailable	870
25 Reference for unit 'Sockets'	871
25.1 Used units	871
25.2 Overview	871
25.3 Constants, types and variables	871
Constants	871
Types	881
Variables	882
25.4 Procedures and functions	882
Accept	882
Bind	884
CloseSocket	884
Connect	885
fpaccept	887
fpbind	887
fpconnect	887
fpgetpeername	887
fpgetsockname	887
fpgetsockopt	887
fplisten	888
fprecv	888
fprecvfrom	888
fpsend	888
fpsendto	888
fpsetsockopt	888
fpshutdown	888
fpsocket	889
fpsocketpair	889
GetPeerName	889
GetSocketName	889
GetSocketOptions	890
htonl	890
htons	890
Listen	891
NToHl	891
NToHs	891
Recv	891

RecvFrom	892
Send	892
SendTo	893
SetSocketOptions	893
Shutdown	894
Sock2File	894
Sock2Text	894
Socket	895
SocketPair	895
Str2UnixSockAddr	895
26 Reference for unit 'strings'	896
26.1 Overview	896
26.2 Procedures and functions	896
stralloc	896
strcat	896
strcomp	897
strcpy	897
strdispose	898
strecopy	899
strend	899
stricomp	900
strlcat	900
strlcomp	901
strlcopy	902
strlen	902
strlicomp	903
strlower	903
strmove	904
strnew	904
strpas	905
strpcopy	905
strpos	906
strrscan	907
strscan	907
strupper	907
27 Reference for unit 'strutils'	908
27.1 Used units	908
27.2 Constants, types and variables	908
Constants	908

Types	909
27.3 Procedures and functions	909
AddChar	909
AddCharR	910
AnsiContainsStr	910
AnsiContainsText	910
AnsiEndsStr	910
AnsiEndsText	911
AnsiIndexStr	911
AnsiIndexText	911
AnsiLeftStr	912
AnsiMatchStr	912
AnsiMatchText	912
AnsiMidStr	912
AnsiProperCase	913
AnsiReplaceStr	913
AnsiReplaceText	913
AnsiResemblesText	914
AnsiReverseString	914
AnsiRightStr	914
AnsiStartsStr	914
AnsiStartsText	915
Copy2Space	915
Copy2SpaceDel	915
Copy2Symb	916
Copy2SymbDel	916
Dec2Numb	916
DecodeSoundexInt	917
DecodeSoundexWord	917
DelChars	917
DelSpace	917
DelSpace1	918
DupeString	918
ExtractDelimited	918
ExtractSubstr	919
ExtractWord	919
ExtractWordPos	919
FindPart	920
GetCmdLineArg	920
Hex2Dec	921

IfThen	921
IntToBin	921
IntToRoman	921
IsEmptyStr	922
IsWild	922
IsWordPresent	922
LeftBStr	923
LeftStr	923
MidBStr	923
MidStr	924
NPos	924
Numb2Dec	924
Numb2USA	925
PadCenter	925
PadLeft	925
PadRight	925
PosEx	926
RandomFrom	926
ReverseString	926
RightBStr	926
RightStr	927
RomanToInt	927
SearchBuf	927
Soundex	928
SoundexCompare	928
SoundexInt	929
SoundexProc	929
SoundexSimilar	929
SoundexWord	930
StuffString	930
Tab2Space	930
WordCount	930
WordPosition	931
XorDecode	931
XorEncode	931
XorString	932
28 Reference for unit 'System'	933
28.1 Miscellaneous functions	933
28.2 Operating System functions	933

28.3 String handling	933
28.4 Mathematical routines	933
28.5 Memory management functions	934
28.6 File handling functions	934
28.7 Variables	934
28.8 Constants	936
28.9 Integer Types	940
28.10 Overview	943
28.11 Constants, types and variables	943
Constants	943
Types	957
Variables	970
28.12 Procedures and functions	972
abs	972
AbstractError	973
AcquireExceptionObject	973
AddExitProc	973
Addr	974
Align	974
AllocMem	974
Append	975
arctan	975
ArrayStringToPPchar	976
AsmFreemem	976
AsmGetmem	976
Assert	977
Assign	977
Assigned	978
binStr	978
BlockRead	979
BlockWrite	980
Break	980
chdir	981
Chr	982
Close	982
CompareByte	983
CompareChar	984
CompareChar0	985
CompareDWord	985
CompareWord	986

Concat	987
Continue	988
copy	989
cos	989
Cseg	990
Dec	990
Delete	991
Dispose	992
Dseg	993
Dump_Stack	993
EOF	993
EOLn	994
Erase	995
Exclude	995
Exit	997
exp	998
FilePos	998
FileSize	999
FillByte	1000
FillChar	1000
FillDWord	1001
FillWord	1002
Flush	1002
frac	1003
Freemem	1003
Freememory	1004
getdir	1004
GetHeapStatus	1005
GetMem	1005
GetMemory	1005
GetMemoryManager	1006
GetProcessID	1006
GetThreadID	1006
GetVariantManager	1006
get_caller_addr	1006
get_caller_frame	1007
get_frame	1007
halt	1007
hexStr	1008
hi	1008

High	1009
Inc	1010
Include	1011
IndexByte	1011
IndexChar	1012
IndexChar0	1013
IndexDWord	1013
Indexword	1014
Insert	1015
int	1015
IOResult	1016
IsMemoryManagerSet	1017
IsVariantManagerSet	1017
length	1017
In	1018
lo	1019
longjmp	1019
Low	1020
lowerCase	1020
MemSize	1021
mkdir	1021
Move	1021
MoveChar0	1022
New	1022
OctStr	1023
odd	1023
Ofs	1024
Ord	1024
Paramcount	1025
ParamStr	1025
pi	1026
Pos	1026
power	1027
Pred	1027
prefetch	1028
ptr	1028
RaiseList	1029
Random	1029
Randomize	1030
Read	1030

ReadLn	1031
Real2Double	1031
ReAllocMem	1032
ReAllocMemory	1032
ReleaseExceptionObject	1032
Rename	1033
Reset	1033
Rewrite	1034
rmdir	1035
round	1035
RunError	1036
Seek	1036
SeekEOF	1037
SeekEOLn	1038
Seg	1038
Setjmp	1039
SetLength	1040
SetMemoryManager	1040
SetMemoryMutexManager	1041
SetString	1041
SetTextBuf	1041
SetTextLineEnding	1042
SetVariantManager	1042
sin	1043
SizeOf	1043
Space	1044
Sptr	1044
sqr	1044
sqrt	1045
Sseg	1045
Str	1046
StringOfChar	1046
StringToPPChar	1047
strlen	1047
strpas	1048
Succ	1048
Swap	1048
SysAllocMem	1049
SysAssert	1049
SysBackTraceStr	1049

SysFreemem	1050
SysFreememSize	1050
SysGetHeapStatus	1050
SysGetmem	1050
SysInitExceptions	1050
SysInitStdIO	1051
SysMemSize	1051
SysReAllocMem	1051
SysResetFPU	1051
SysTryResizeMem	1051
trunc	1052
Truncate	1052
UniqueString	1053
upCase	1053
Val	1054
Write	1054
WriteLn	1055
28.13 TObject	1055
Description	1055
Method overview	1056
TObject.Create	1056
TObject.Destroy	1056
TObject.newinstance	1057
TObject.FreeInstance	1057
TObject.SafeCallException	1057
TObject.DefaultHandler	1057
TObject.Free	1058
TObject.InitInstance	1058
TObject.CleanupInstance	1058
TObject.ClassType	1059
TObject.ClassInfo	1059
TObject.ClassName	1059
TObject.ClassNameIs	1059
TObject.ClassParent	1060
TObject.InstanceSize	1060
TObject.InheritsFrom	1060
TObject.StringMessageTable	1060
TObject.Dispatch	1061
TObject.DispatchStr	1061
TObject.MethodAddress	1061

TObject.MethodName	1061
TObject.FieldAddress	1062
TObject.AfterConstruction	1062
TObject.BeforeDestruction	1062
TObject.DefaultHandlerStr	1062
29 Reference for unit 'systhds'	1064
29.1 Used units	1064
29.2 Constants, types and variables	1064
Constants	1064
Types	1064
29.3 Procedures and functions	1066
BasicEventCreate	1066
basiceventdestroy	1067
basiceventResetEvent	1067
basiceventSetEvent	1067
basiceventWaitFor	1067
BeginThread	1067
DoneCriticalsection	1067
EndThread	1067
EnterCriticalsection	1067
GetCurrentThreadId	1068
GetThreadManager	1068
InitCriticalSection	1068
InitThread	1068
KillThread	1068
LeaveCriticalSection	1068
ResumeThread	1068
RTLEventCreate	1068
RTLeventdestroy	1068
RTLeventSetEvent	1069
RTLeventsync	1069
RTLeventWaitFor	1069
SetNoThreadManager	1069
SetThreadManager	1069
SuspendThread	1069
ThreadGetPriority	1069
ThreadSetPriority	1069
ThreadSwitch	1070
WaitForThreadTerminate	1070

30 Reference for unit 'sysutils'	1071
30.1 Miscellaneous conversion routines	1071
30.2 Date/time routines	1071
30.3 FileName handling routines	1071
30.4 File input/output routines	1071
30.5 PChar related functions	1072
30.6 Date and time formatting characters	1074
30.7 Formatting strings	1075
30.8 String functions	1075
30.9 Used units	1075
30.10 Overview	1075
30.11 Constants, types and variables	1075
Constants	1075
Types	1082
Variables	1087
30.12 Procedures and functions	1088
Abort	1088
AddDisk	1088
AddTerminateProc	1089
AdjustLineBreaks	1089
AnsiCompareFileName	1090
AnsiCompareStr	1090
AnsiCompareText	1091
AnsiExtractQuotedStr	1092
AnsiLastChar	1092
AnsiLowerCase	1093
AnsiLowerCaseFileName	1094
AnsiPos	1094
AnsiQuotedStr	1094
AnsiSameStr	1095
AnsiSameText	1095
AnsiStrComp	1095
AnsiStrIComp	1096
AnsiStrLastChar	1097
AnsiStrLComp	1098
AnsiStrLIComp	1098
AnsiStrLower	1099
AnsiStrPos	1100
AnsiStrRScan	1100
AnsiStrScan	1100

AnsiStrUpper	1100
AnsiUpperCase	1101
AnsiUpperCaseFileName	1102
AppendStr	1102
ApplicationName	1103
AssignStr	1103
BCDToInt	1103
Beep	1104
BoolToStr	1104
ByteToCharIndex	1104
ByteToCharLen	1105
ByteType	1105
CallTerminateProcs	1105
ChangeFileExt	1106
CharToByteLen	1106
CompareMem	1106
CompareMemRange	1107
CompareStr	1107
CompareText	1108
CreateDir	1109
CurrToStr	1109
Date	1110
DateTimeToFileDate	1110
DateTimeToStr	1111
DateTimeToString	1111
DateTimeToSystemTime	1112
DateTimeToTimeStamp	1113
DateToStr	1113
DayOfWeek	1114
DecodeDate	1114
DecodeDateFully	1115
DecodeTime	1115
DeleteFile	1116
DirectoryExists	1116
DiskFree	1117
DiskSize	1117
DisposeStr	1118
DoDirSeparators	1118
EncodeDate	1119
EncodeTime	1119

ExceptAddr	1120
ExceptionErrorMessage	1120
ExceptObject	1121
ExcludeTrailingBackslash	1121
ExcludeTrailingPathDelimiter	1121
ExecuteProcess	1121
ExpandFileName	1122
ExpandUNCFileName	1122
ExtractFileDir	1123
ExtractFileDrive	1123
ExtractFileExt	1124
ExtractFileName	1124
ExtractFilePath	1124
ExtractRelativepath	1125
FileAge	1125
FileClose	1126
FileCreate	1126
FileDateToDateTime	1127
FileExists	1128
FileGetAttr	1128
FileGetDate	1129
FileIsReadOnly	1130
FileOpen	1130
FileRead	1131
FileSearch	1131
FileSeek	1132
FileSetAttr	1133
FileSetDate	1133
FileTruncate	1133
FileWrite	1133
FindClose	1134
FindCmdLineSwitch	1134
FindFirst	1134
FindNext	1135
FloattoCurr	1136
FloatToDateTime	1136
FloatToDecimal	1136
FloatToStr	1137
FloatToStrF	1137
FloatToText	1139

FloatToTextFmt	1140
FmtStr	1140
Format	1141
FormatBuf	1146
FormatDateTime	1147
FormatFloat	1147
FreeAndNil	1149
GetAppConfigDir	1149
GetAppConfigFile	1150
GetCurrentDir	1150
GetDirs	1151
GetEnvironmentString	1151
GetEnvironmentVariable	1152
GetEnvironmentVariableCount	1152
GetFileHandle	1152
GetLastError	1153
GetLocalTime	1153
GetTempDir	1153
GetTempFileName	1154
IncludeTrailingBackslash	1154
IncludeTrailingPathDelimiter	1154
IncMonth	1154
InterLockedDecrement	1155
InterLockedExchange	1155
InterLockedExchangeAdd	1156
InterLockedIncrement	1156
IntToHex	1156
IntToStr	1157
IsDelimiter	1157
IsLeapYear	1158
IsPathDelimiter	1158
IsValidIdent	1158
LastDelimiter	1159
LeftStr	1160
LoadStr	1160
LowerCase	1160
MSecsToTimeStamp	1161
NewStr	1161
Now	1162
OutOfMemoryError	1162

QuotedStr	1163
RaiseLastError	1163
RemoveDir	1163
RenameFile	1164
RightStr	1164
SameFileName	1165
SameText	1165
SetCurrentDir	1165
SetDirSeparators	1166
ShowException	1166
Sleep	1166
StrAlloc	1167
StrBufSize	1167
StrByteType	1168
strcat	1168
strcmp	1168
strcpy	1169
StrDispose	1169
strecopy	1170
strend	1170
StrFmt	1171
stricmp	1171
StringReplace	1172
strlcat	1172
strlcomp	1173
strlcopy	1174
strlen	1174
StrLFmt	1175
strlicomp	1175
strlower	1176
strmove	1176
strnew	1177
StrPas	1177
StrPCopy	1178
StrPLCopy	1178
strpos	1178
strscan	1179
strscan	1179
StrToBool	1179
StrToCurr	1180

StrToCurrDef	1180
StrToDate	1180
StrToDateTime	1181
StrToFloat	1182
StrToFloatDef	1183
StrToInt	1183
StrToInt64	1184
StrToInt64Def	1184
StrToIntDef	1184
StrToTime	1185
strupper	1186
SysErrorMessage	1186
SystemTimeToDateTime	1186
TextToFloat	1187
Time	1188
TimeStampToDateTime	1188
TimeStampToMsecs	1189
TimeToStr	1189
Trim	1190
TrimLeft	1190
TrimRight	1191
TryEncodeDate	1191
TryEncodeTime	1192
TryFloatToCurr	1192
TryStrToInt	1192
TryStrToInt64	1193
UpperCase	1193
WrapText	1193
30.13EAbort	1194
Description	1194
30.14EAbstractError	1194
Description	1194
30.15EAccessViolation	1194
Description	1194
30.16EAssertionFailed	1194
Description	1194
30.17EControlC	1194
Description	1194
30.18EConvertError	1194
Description	1194

30.19EDivByZero	1194
Description	1194
30.20EExternal	1195
Description	1195
30.21EExternalException	1195
Description	1195
30.22EHeapMemoryError	1195
Description	1195
Method overview	1195
EHeapMemoryError.FreeInstance	1195
30.23EInOutError	1195
Description	1195
30.24EIntError	1195
Description	1195
30.25EIntfCastError	1196
Description	1196
30.26EIntOverflow	1196
Description	1196
30.27EInvalidCast	1196
Description	1196
30.28EInvalidContainer	1196
Description	1196
30.29EInvalidInsert	1196
Description	1196
30.30EInvalidOp	1196
Description	1196
30.31EInvalidPointer	1196
Description	1196
30.32EMathError	1197
Description	1197
30.33ENoThreadSupport	1197
Description	1197
30.34EOSError	1197
Description	1197
30.35EOOutOfMemory	1197
Description	1197
30.36EOOverflow	1197
Description	1197
30.37EPackageError	1197
Description	1197

30.38EPrivilege	1197
Description	1197
30.39EPropReadOnly	1198
Description	1198
30.40EPropWriteOnly	1198
Description	1198
30.41ERangeError	1198
Description	1198
30.42ESafecallException	1198
Description	1198
30.43EStackOverflow	1198
Description	1198
30.44EUnderflow	1198
Description	1198
30.45EVariantError	1198
Description	1198
30.46Exception	1199
Description	1199
Method overview	1199
Property overview	1199
Exception.Create	1199
Exception.CreateFmt	1199
Exception.CreateRes	1200
Exception.CreateResFmt	1200
Exception.CreateHelp	1200
Exception.CreateFmtHelp	1200
Exception.CreateResHelp	1201
Exception.CreateResFmtHelp	1201
Exception.HelpContext	1201
Exception.Message	1201
30.47EZeroDivide	1202
Description	1202
31 Reference for unit 'typinfo'	1203
31.1 Auxiliary functions	1203
31.2 Getting or setting property values	1203
31.3 Examining published property information	1203
31.4 Used units	1203
31.5 Overview	1203
31.6 Constants, types and variables	1204

Constants	1204
Types	1205
31.7 Procedures and functions	1208
FindPropInfo	1208
GetEnumName	1210
GetEnumProp	1211
GetEnumValue	1212
GetFloatProp	1212
GetInt64Prop	1213
GetMethodProp	1214
GetObjectProp	1216
GetObjectPropClass	1217
GetOrdProp	1217
GetPropInfo	1218
GetPropInfos	1219
GetPropList	1220
GetPropValue	1221
GetSetProp	1221
GetStrProp	1223
GetTypeData	1223
GetVariantProp	1224
IsPublishedProp	1224
IsStoredProp	1225
PropIsType	1226
PropType	1226
SetEnumProp	1227
SetFloatProp	1228
SetInt64Prop	1228
SetMethodProp	1228
SetObjectProp	1229
SetOrdProp	1229
SetPropValue	1230
SetSetProp	1230
SetStrProp	1231
SetToString	1231
SetVariantProp	1232
StringToSet	1232
31.8 EPropertyError	1233
Description	1233

32 Reference for unit 'Unix'	1234
32.1 Used units	1234
32.2 Constants, types and variables	1234
Constants	1234
Types	1244
Variables	1252
32.3 Procedures and functions	1252
AssignPipe	1252
AssignStream	1253
FpExecL	1254
FpExecLE	1255
FpExecLP	1256
FpExecV	1257
FpExecVP	1258
FpExecVPE	1259
fpFlock	1260
fpgettimeofday	1260
fpSystem	1261
FSearch	1261
fStatFS	1262
fsync	1263
GetDomainName	1263
GetHostName	1264
GetLocalTimezone	1264
GetTimezoneFile	1265
PClose	1265
POpen	1265
ReadTimezoneFile	1266
SeekDir	1267
SelectText	1267
Shell	1267
SigRaise	1268
StatFS	1269
TellDir	1270
WaitProcess	1270
WIFSTOPPED	1271
W_EXITCODE	1271
W_STOPCODE	1271
33 Reference for unit 'unixtype'	1272

33.1 Overview	1272
33.2 Constants, types and variables	1272
Constants	1272
Types	1273
34 Reference for unit 'unixutil'	1282
34.1 Overview	1282
34.2 Constants, types and variables	1282
Types	1282
Variables	1282
34.3 Procedures and functions	1283
ArrayStringToPPchar	1283
Basename	1283
Dirname	1284
EpochToLocal	1284
FNMatch	1285
FSplit	1286
GetFS	1286
GregorianToJulian	1287
JulianToGregorian	1287
LocalToEpoch	1287
StringToPPChar	1288
35 Reference for unit 'Video'	1290
35.1 Examples utility unit	1290
35.2 Writing a custom video driver	1290
35.3 Overview	1291
35.4 Constants, types and variables	1292
Constants	1292
Types	1295
Variables	1297
35.5 Procedures and functions	1297
ClearScreen	1297
DefaultErrorHandler	1298
DoneVideo	1298
GetCapabilities	1299
GetCursorType	1300
GetLockScreenCount	1301
GetVideoDriver	1301
GetVideoMode	1302
GetVideoModeCount	1302

GetVideoModeData	1304
InitVideo	1304
LockScreenUpdate	1304
SetCursorPos	1305
SetCursorType	1306
SetVideoDriver	1306
SetVideoMode	1307
UnlockScreenUpdate	1307
UpdateScreen	1307
36 Reference for unit 'x86'	1309
36.1 Used units	1309
36.2 Overview	1309
36.3 Procedures and functions	1309
fpIOperm	1309
fpIoPL	1310
ReadPort	1310
ReadPortB	1310
ReadPortL	1311
ReadPortW	1311
WritePort	1311
WritePortB	1312
WritePortl	1312
WritePortW	1312

About this guide

This document describes all constants, types, variables, functions and procedures as they are declared in the units that come standard with the FCL (Free Component Library).

Throughout this document, we will refer to functions, types and variables with `typewriter` font. Functions and procedures have their own subsections, and for each function or procedure we have the following topics:

Declaration The exact declaration of the function.

Description What does the procedure exactly do ?

Errors What errors can occur.

See Also Cross references to other related functions/commands.

0.1 Overview

The Run-Time Library is the basis of all Free Pascal programs. It contains the basic units that most programs will use, and are made available on all platforms supported by Free pascal (well, more or less).

There are units for compatibility with the Turbo Pascal Run-Time library, and there are units for compatibility with Delphi.

On top of these two sets, there are also a series of units to handle keyboard/mouse and text screens in a cross-platform way.

Other units include platform specific units that implement the specifics of a platform, these are usually needed to support the Turbo Pascal or Delphi units.

Units that fall outside the above outline do not belong in the RTL, but should be included in the packages, or in the FCL.

Chapter 1

Reference for unit 'BaseUnix'

1.1 Used units

Table 1.1: Used units by unit 'BaseUnix'

Name	Page
unixtype	1272

1.2 Overview

The **BaseUnix** unit was implemented by Marco Van de Voort. It contains basic unix functionality. It supersedes the **Linux** unit of version 1.0.X of the compiler, but does not implement all functionality of the **linux** unit.

People that have code which heavily uses the old **Linux** unit, can simply change **linux** by **oldlinux** in the **uses** clause of their projects, but they should really consider moving to the **Unix** and **BaseUnix** units.

For porting FPC to new unix-like platforms, it should be sufficient to implement the functionality in this unit for the new platform.

1.3 Constants, types and variables

Constants

`ARG_MAX = UnixType.ARG_MAX`

Maximum number of arguments to a program.

`BITSINWORD = 8 * sizeof (longint)`

Number of bits in a word.

`ESysE2BIG = 7`

System error: Argument list too long

ESysEACCES = 13

System error: Permission denied

ESysEADDRINUSE = 98

System error: Address already in use

ESysEADDRNOTAVAIL = 99

System error: Cannot assign requested address

ESysEADV = 68

System error: Advertise error

ESysEAFNOSUPPORT = 97

System error: Address family not supported by protocol

ESysEAGAIN = 11

System error: Try again

ESysEALREADY = 114

System error: Operation already in progress

ESysEBADE = 52

System error: Invalid exchange

ESysEBADF = 9

System error: Bad file number

ESysEBADFD = 77

System error: File descriptor in bad state

ESysEBADMSG = 74

System error: Not a data message

ESysEBADR = 53

System error: Invalid request descriptor

ESysEBADRQC = 56

System error: Invalid request code

ESysEBADSLT = 57

System error: Invalid slot

ESysEBFONT = 59

System error: Bad font file format

ESysEBUSY = 16

System error: Device or resource busy

ESysECHILD = 10

System error: No child processes

ESysECHRNG = 44

System error: Channel number out of range

ESysECOMM = 70

System error: Communication error on send

ESysECONNABORTED = 103

System error: Software caused connection abort

ESysECONNREFUSED = 111

System error: Connection refused

ESysECONNRESET = 104

System error: Connection reset by peer

ESysEDEADLK = 35

System error: Resource deadlock would occur

ESysEDEADLOCK = 58

System error: File locking deadlock error

ESysEDESTADDRREQ = 89

System error: Destination address required

ESysEDOM = 33

System error: Math argument out of domain of func

ESysEDOTDOT = 73

System error: RFS specific error

ESysEDQUOT = 122

System error: Quota exceeded

ESysEEXIST = 17

System error: File exists

ESysEFAULT = 14

System error: Bad address

ESysEFBIG = 27

System error: File too large

ESysEHOSTDOWN = 112

System error: Host is down

ESysEHOSTUNREACH = 113

System error: No route to host

ESysEIDRM = 43

System error: Identifier removed

ESysEILSEQ = 84

System error: Illegal byte sequence

ESysEINPROGRESS = 115

System error: Operation now in progress

ESysEINTR = 4

System error: Interrupted system call

ESysEINVAL = 22

System error: Invalid argument

ESysEIO = 5

System error: I/O error

ESysEISCONN = 106

System error: Transport endpoint is already connected

ESysEISDIR = 21

System error: Is a directory

ESysEISNAM = 120

System error: Is a named type file

ESysEL2HLT = 51

System error: Level 2 halted

ESysEL2NSYNC = 45

System error: Level 2 not synchronized

ESysEL3HLT = 46

System error: Level 3 halted

ESysEL3RST = 47

System error: Level 3 reset

ESysELIBACC = 79

System error: Can not access a needed shared library

ESysELIBBAD = 80

System error: Accessing a corrupted shared library

ESysELIBEXEC = 83

System error: Cannot exec a shared library directly

ESysELIBMAX = 82

System error: Attempting to link in too many shared libraries

ESysELIBSCN = 81

System error: .lib section in a.out corrupted

ESysELNRNG = 48

System error: Link number out of range

ESysELOOP = 40

System error: Too many symbolic links encountered

ESysEMFILE = 24

System error: Too many open files

ESysEMLINK = 31

System error: Too many links

ESysEMSGSIZE = 90

System error: Message too long

ESysEMULTIHOP = 72

System error: Multihop attempted

ESysENAMETOOLONG = 36

System error: File name too long

ESysENAVAIL = 119

System error: No XENIX semaphores available

ESysENETDOWN = 100

System error: Network is down

ESysENETRESET = 102

System error: Network dropped connection because of reset

ESysENETUNREACH = 101

System error: Network is unreachable

ESysENFILE = 23

System error: File table overflow

ESysENOANO = 55

System error: No anode

ESysENOBUFFS = 105

System error: No buffer space available

ESysENOCSSI = 50

System error: No CSI structure available

ESysENODATA = 61

System error: No data available

ESysENODEV = 19

System error: No such device

ESysENOENT = 2

System error: No such file or directory

ESysENOEXEC = 8

System error: Exec format error

ESysENOLCK = 37

System error: No record locks available

ESysENOLINK = 67

System error: Link has been severed

ESysENOMEM = 12

System error: Out of memory

ESysENOMSG = 42

System error: No message of desired type

ESysENONET = 64

System error: Machine is not on the network

ESysENOPKG = 65

System error: Package not installed

ESysENOPROTOOPT = 92

System error: Protocol not available

ESysENOSPC = 28

System error: No space left on device

ESysENOSR = 63

System error: Out of streams resources

ESysENOSTR = 60

System error: Device not a stream

ESysENOSYS = 38

System error: Function not implemented

ESysENOTBLK = 15

System error: Block device required

ESysENOTCONN = 107

System error: Transport endpoint is not connected

ESysENOTDIR = 20

System error: Not a directory

ESysENOTEMPTY = 39

System error: Directory not empty

ESysENOTNAM = 118

System error: Not a XENIX named type file

ESysENOTSOCK = 88

System error: Socket operation on non-socket

ESysENOTTY = 25

System error: Not a typewriter

ESysENOTUNIQ = 76

System error: Name not unique on network

ESysENXIO = 6

System error: No such device or address

ESysEOPNOTSUPP = 95

System error: Operation not supported on transport endpoint

ESysEOVERFLOW = 75

System error: Value too large for defined data type

ESysEPERM = 1

System error: Operation not permitted.

ESysEPFNOSUPPORT = 96

System error: Protocol family not supported

ESysEPIPE = 32

System error: Broken pipe

ESysEPROTO = 71

System error: Protocol error

ESysEPROTONOSUPPORT = 93

System error: Protocol not supported

ESysEPROTOTYPE = 91

System error: Protocol wrong type for socket

ESysERANGE = 34

System error: Math result not representable

ESysEREMCHG = 78

System error: Remote address changed

ESysEREMOTE = 66

System error: Object is remote

ESysEREMOTEIO = 121

System error: Remote I/O error

ESysERESTART = 85

System error: Interrupted system call should be restarted

ESysEROFS = 30

System error: Read-only file system

ESysESHUTDOWN = 108

System error: Cannot send after transport endpoint shutdown

ESysESOCKTNOSUPPORT = 94

System error: Socket type not supported

ESysESPIPE = 29

System error: Illegal seek

ESysESRCH = 3

System error: No such process

ESysESRMNT = 69

System error: Srmount error

ESysESTALE = 116

System error: Stale NFS file handle

ESysESTRPIPE = 86

System error: Streams pipe error

ESysETIME = 62

System error: Timer expired

ESysETIMEDOUT = 110

System error: Connection timed out

ESysETOOMANYREFS = 109

System error: Too many references: cannot splice

ESysETXTBSY = 26

System error: Text (code segment) file busy

ESysEUCLEAN = 117

System error: Structure needs cleaning

ESysEUNATCH = 49

System error: Protocol driver not attached

ESysEUSERS = 87

System error: Too many users

ESysEWOULDBLOCK = ESysEAGAIN

System error: Operation would block

ESysEXDEV = 18

System error: Cross-device link

ESysEXFULL = 54

System error: Exchange full

FD_MAXFDSET = 1024

Maximum elements in a TFDSet (95) array.

F_GetFd = 1

fpFCntl (108) command: Get close-on-exec flag

F_GetFl = 3

fpFCntl (108) command: Get filedescriptor flags

F_GetLk = 5

fpFCntl (108) command: Get lock

F_GetOwn = 9

fpFCntl (108) command: get owner of filedescriptor events

F_OK = 0

fpAccess (99) call test: file exists.

F_SetFd = 2

fpFCntl (108) command: Set close-on-exec flag

F_SetFl = 4

fpFCntl (108) command: Set filedescriptor flags

F_SetLk = 6

fpFCntl (108) command: Set lock

F_SetLkW = 7

fpFCntl (108) command: Test lock

F_SetOwn = 8

fpFCntl (108) command: Set owner of filedescriptor events

ln2bitmask = 1 shl ln2bitsinword - 1

Last bit in word.

ln2bitsinword = 5

Power of 2 number of bits in word.

NAME_MAX = UnixType.NAME_MAX

Maximum filename length.

O_APPEND = \$400

fpOpen (124) file open mode: Append to file

O_CREAT = \$40

fpOpen (124) file open mode: Create if file does not yet exist.

O_DIRECT = \$4000

fpOpen (124) file open mode: Minimize caching effects

O_DIRECTORY = \$10000

fpOpen (124) file open mode: File must be directory.

O_EXCL = \$80

fpOpen (124) file open mode: Open exclusively

O_LARGEFILE = \$8000

fpOpen (124) file open mode: Open for 64-bit I/O

O_NDELAY = O_NONBLOCK

fpOpen (124) file open mode: Alias for O_NonBlock (82)

O_NOCTTY = \$100

fpOpen (124) file open mode: No TTY control.

O_NOFOLLOW = \$20000

fpOpen (124) file open mode: Fail if file is symbolic link.

O_NONBLOCK = \$800

fpOpen (124) file open mode: Open in non-blocking mode

O_RDONLY = 0

fpOpen (124) file open mode: Read only

O_RDWR = 2

fpOpen (124) file open mode: Read/Write

O_SYNC = \$1000

fpOpen (124) file open mode: Write to disc at once

O_TRUNC = \$200

fpOpen (124) file open mode: Truncate file to length 0

O_WRONLY = 1

fpOpen (124) file open mode: Write only

PATH_MAX = UnixType.PATH_MAX

Maximum pathname length.

R_OK = 4

fpAccess (99) call test: read allowed

SA_INTERRUPT = \$20000000

Sigaction options: ?

SA_NOCLDSTOP = 1

Sigaction options: Do not receive notification when child processes stop

SA_NOCLDWAIT = 2

Sigaction options: ?

SA_NOMASK = \$40000000

Sigaction options: Do not prevent the signal from being received when it is handled.

SA_ONESHOT = \$80000000

Sigaction options: Restore the signal action to the default state.

SA_RESTART = \$10000000

Sigaction options: Provide behaviour compatible with BSD signal semantics

SA_SHIRQ = \$04000000

Sigaction options: ?

SA_SIGINFO = 4

Sigaction options: The signal handler takes 3 arguments, not one.

SA_STACK = \$08000000

Sigaction options: Call the signal handler on an alternate signal stack.

SEEK_CUR = 1

fpLSeek (118) option: Set position relative to current position.

SEEK_END = 2

fpLSeek (118) option: Set position relative to end of file.

SEEK_SET = 0

fpLSeek (118) option: Set absolute position.

SIGABRT = 6

Signal: ABRT (Abort)

SIGALRM = 14

Signal: ALRM (Alarm clock)

SIGBUS = 7

Signal: BUS (bus error)

SIGCHLD = 17

Signal: CHLD (child status changed)

SIGCONT = 18

Signal: CONT (Continue)

SIGFPE = 8

Signal: FPE (Floating point error)

SIGHUP = 1

Signal: HUP (Hangup)

SIGILL = 4

Signal: ILL (Illegal instruction)

SIGINT = 2

Signal: INT (Interrupt)

SIGIO = 29

Signal: IO (I/O operation possible)

SIGIOT = 6

Signal: IOT (IOT trap)

SIGKILL = 9

Signal: KILL (unblockable)

SIGPIPE = 13

Signal: PIPE (Broken pipe)

SIGPOLL = SIGIO

Signal: POLL (Pollable event)

SIGPROF = 27

Signal: PROF (Profiling alarm)

SIGPWR = 30

Signal: PWR (power failure restart)

SIGQUIT = 3

Signal: QUIT

SIGSEGV = 11

Signal: SEGV (Segmentation violation)

SIGSTKFLT = 16

Signal: STKFLT (Stack Fault)

SIGSTOP = 19

Signal: STOP (Stop, unblockable)

SIGTerm = 15

Signal: TERM (Terminate)

SIGTRAP = 5

Signal: TRAP (Trace trap)

SIGTSTP = 20

Signal: TSTP (keyboard stop)

SIGTTIN = 21

Signal: TTIN (Terminal input, background)

SIGTTOU = 22

Signal: TTOU (Terminal output, background)

SIGUNUSED = 31

Signal: Unused

SIGURG = 23

Signal: URG (Socket urgent condition)

SIGUSR1 = 10

Signal: USR1 (User-defined signal 1)

SIGUSR2 = 12

Signal: USR2 (User-defined signal 2)

SIGVTALRM = 26

Signal: VTALRM (Virtual alarm clock)

SIGWINCH = 28

Signal: WINCH (Window/Terminal size change)

SIGXCPU = 24

Signal: XCPU (CPU limit exceeded)

SIGXFSZ = 25

Signal: XFSZ (File size limit exceeded)

SIG_BLOCK = 0

Sigprocmask flags: Add signals to the set of blocked signals.

SIG_DFL = 0

Signal handler: Default signal handler

SIG_ERR = -1

Signal handler: error

SIG_IGN = 1

Signal handler: Ignore signal

SIG_MAXSIG = UnixType.SIG_MAXSIG

Maximum system signal number.

SIG_SETMASK = 2

Sigprocmask flags: Set of blocked signals is given.

SIG_UNBLOCK = 1

Sigprocmask flags: Remove signals from the set set of blocked signals.

SI_PAD_SIZE = ((128 div sizeof (longint)) - 3)

Signal information pad size.

SYS_NMLN = UnixType.SYS_NMLN

Max system name length.

S_IFBLK = 24576

S_IFCHR = 8192

S_IFDIR = 16384

S_IFIFO = 4096

S_IFLNK = 40960

S_IFMT = 61440

S_IFREG = 32768

S_IFSOCK = 49152

S_IRGRP = %0000100000

Mode flag: Read by group.

S_IROTH = %0000000100

Mode flag: Read by others.

S_IRUSR = %0100000000

Mode flag: Read by owner.

S_IWGRP = %0000010000

Mode flag: Write by group.

S_IWOTH = %0000000010

Mode flag: Write by others.

S_IWUSR = %0010000000

Mode flag: Write by owner.

S_IXGRP = %0000001000

Mode flag: Execute by group.

S_IXOTH = %0000000001

Mode flag: Execute by others.

S_IXUSR = %0001000000

Mode flag: Execute by owner.

UTSNAME_DOMAIN_LENGTH = UTSNAME_LENGTH

Max length of `utsname` (98) domain name.

UTSNAME_LENGTH = SYS_NMLN

Max length of `utsname` (98) system name, release, version, machine.

UTSNAME_NODENAME_LENGTH = UTSNAME_LENGTH

Max length of `utsname` (98) node name.

`WNOHANG` = 1

`#rtl.baseunix.fpWaitpid` (147) option: Do not wait for processes to terminate.

`wordsinfdset` = `FD_MAXFDSET` div `BITSINWORD`

Number of words in a `TFDSet` (95) array

`wordsinsigset` = `SIG_MAXSIG` div `BITSINWORD`

Number of words in a signal set.

`WUNTRACED` = 2

`#rtl.baseunix.fpWaitpid` (147) option: Also report children which were stopped but not yet reported

`W_OK` = 2

`fpAccess` (99) call test: write allowed

`X_OK` = 1

`fpAccess` (99) call test: execute allowed

Types

`Blkcnt_t` = `cUInt`

Block count type.

`Blksize_t` = `cUInt`

Block size type.

`cInt` = `UnixType.cInt`

C type: integer (natural size)

`cInt16` = `UnixType.cInt16`

C type: 16 bits sized, signed integer.

`cInt32` = `UnixType.cInt32`

C type: 32 bits sized, signed integer.

`cInt64` = `UnixType.cInt64`

C type: 64 bits sized, signed integer.

`cInt8` = `UnixType.cInt8`

C type: 8 bits sized, signed integer.

`clock_t = UnixType.clock_t`

Clock ticks type

`cLong = UnixType.cLong`

C type: long signed integer (double sized)

`cshort = UnixType.cshort`

C type: short signed integer (half sized)

`cuchar = UnixType.cuchar`

C type: unsigned character

`cUInt = UnixType.cUInt`

C type: unsigned integer (natural size)

`cUInt16 = UnixType.cUInt16`

C type: 16 bits sized, unsigned integer.

`cUInt32 = UnixType.cUInt32`

C type: 32 bits sized, unsigned integer.

`cUInt64 = UnixType.cUInt64`

C type: 64 bits sized, unsigned integer.

`cUInt8 = UnixType.cUInt8`

C type: 8 bits sized, unsigned integer.

`cuLong = UnixType.cuLong`

C type: long unsigned integer (double sized)

`cushort = UnixType.cushort`

C type: short unsigned integer (half sized)

`dev_t = UnixType.dev_t`

Device descriptor type.

```
Dir = record
  dd_fd : Integer;
  dd_loc : LongInt;
  dd_size : Integer;
  dd_buf : pDirent;
  dd_nextoff : LongInt;
  dd_max : Integer;
  dd_lock : pointer;
end
```

Record used in `fpOpenDir` (125) and `fpReadDir` (129) calls

```
Dirent = packed record
  d_fileno : ino_t;
  d_off : off_t;
  d_reclen : cushort;
  d_name : Array[0..(255+1)-1] of Char;
end
```

Record used in the `fpReadDir` (129) function to return files in a directory.

```
Flock = record
  l_type : cshort;
  l_whence : cshort;
  l_start : off_t;
  l_len : off_t;
  l_pid : pid_t;
end
```

Lock description type for `fpFCntl` (108) lock call.

```
gid_t = UnixType.gid_t
```

Group ID type.

```
Ino64_t = cInt64
```

64-bit inode type.

```
ino_t = UnixType.ino_t
```

Inode type.

```
mode_t = UnixType.mode_t
```

Inode mode type.

```
nlink_t = UnixType.nlink_t
```

Number of links type.

```
Off64_t = cInt64
```

64-bit offset type.

```
off_t = UnixType.off_t
```

Offset type.

```
PBlkCnt = ^Blkcnt_t
```

pointer to `TBlkCnt` (95) type.

`PBlkSize = ^Blksize_t`

Pointer to TBlkSize (95) type.

`pcInt = UnixType.pcInt`

Pointer to cInt (88) type.

`pClock = UnixType.pClock`

Pointer to TClock (95) type.

`pcLong = UnixType.pcLong`

Pointer to cLong (89) type.

`pcshort = UnixType.pcsshort`

Pointer to cShort (89) type.

`pcUInt = UnixType.pcUInt`

Pointer to cUInt (89) type.

`pculong = UnixType.pculong`

Pointer to cuLong (89) type.

`pcushort = UnixType.pcushort`

Pointer to cuShort (89) type.

`pDev = UnixType.pDev`

Pointer to TDev (95) type.

`pDir = ^Dir`

Pointer to TDir (95) record

`pDirent = ^Dirent`

Pointer to TDirent (95) record.

`pFDSet = ^TFDSet`

Pointer to TFDSet (95) type.

`pFilDes = ^TFilDes`

Pointer to TFilDes (95) type.

`pfpstate = ^tfpstate`

Pointer to tfpstate (96) record.

`pGid = UnixType.pGid`

Pointer to TGid (96) type.

`pGrpArr = ^TGrpArr`

Pointer to TGrpArr (96) array.

`pid_t = UnixType.pid_t`

Process ID type.

`pIno = UnixType.pIno`

Pointer to TIno (96) type.

`PIno64 = ^Ino64_t`

Pointer to TIno64 (96) type.

`pMode = UnixType.pMode`

Pointer to TMode (96) type.

`pnLink = UnixType.pnLink`

Pointer to TnLink (97) type.

`pOff = UnixType.pOff`

Pointer to TOff (97) type.

`POff64 = ^Off64_t`

Pointer to TOff64 (97) type.

`pPid = UnixType.pPid`

Pointer to TPid (97) type.

`PSigActionRec = ^SigActionRec`

Pointer to SigActionRec (94) record type.

`PSigInfoRec = ^SigInfoRec`

Pointer to SigInfoRec (94) record type.

`PSignalHandler = ^SignalHandler`

Pointer to SignalHandler (94) type.

`PSignalRestorer = ^SignalRestorer`

Pointer to SignalRestorer (94) type

`PSigSet = ^SigSet`

Pointer to `SigSet` (94) type.

`psigset_t = PSigSet`

Pointer to `sigset_t` (94) type.

`pSize = UnixType.pSize`

Pointer to `TSize` (97) type.

`pSocklen = UnixType.pSocklen`

Pointer to `TSockLen` (97) type.

`psSize = UnixType.psSize`

Pointer to `TsSize` (97) type

`PStat = ^Stat`

Pointer to `TStat` (97) type.

`pthread_cond_t = UnixType.pthread_cond_t`

Thread conditional variable type.

`pthread_mutex_t = UnixType.pthread_mutex_t`

Thread mutex type.

`pthread_t = UnixType.pthread_t`

Posix thread type.

`pTime = UnixType.pTime`

Pointer to `TTime` (98) type.

`ptimespec = UnixType.ptimespec`

Pointer to `timespec` (96) type.

`ptimeval = UnixType.ptimeval`

Pointer to `timeval` (96) type.

`ptimezone = ^timezone`

`ptime_t = UnixType.ptime_t`

Pointer to `time_t` (96) type.

PTms = ^tms

Pointer to TTms (98) type.

pUId = UnixType.pUId

Pointer to TUid (98) type.

pUtimBuf = ^UTimBuf

Pointer to TUTimBuf (98) type.

PUtsName = TUtilsName

Pointer to TUtilsName (98) type.

```
SigActionRec = packed record
  sa_handler : SignalHandler;
  Sa_Mask : SigSet;
  Sa_Flags : LongInt;
  Sa_restorer : SignalRestorer;
end
```

Record used in fpSigAction (134) call.

```
SigInfoRec = record
  si_signo : LongInt;
  si_errno : LongInt;
  si_code : LongInt;
end
```

Record containing signal information.

SignalHandler = procedure(Sig: LongInt)

Simple signal handler prototype

SignalRestorer = procedure

Signal restorer function prototype

SigSet = Array[0..wordsinsigset-1] of cInt

Signal set type

sigset_t = SigSet

Signal set type

size_t = UnixType.size_t

Size specification type.

`socklen_t = UnixType.socklen_t`

Socket address length type.

`ssize_t = UnixType.ssize_t`

Small size type.

`Stat = packed record`
`end`

Record describing an inode (file) in the `FPFstat` (110) call.

`TBlkCnt = Blkcnt_t`

Alias for `Blkcnt_t` (88) type.

`TBlkSize = Blksize_t`

Alias for `blksize_t` (88) type.

`TClock = UnixType.TClock`

Alias for `clock_t` (89) type.

`TDev = UnixType.TDev`

Alias for `dev_t` (89) type.

`TDDir = Dir`

Alias for `Dir` (90) type.

`TDirent = Dirent`

Alias for `Dirent` (90) type.

`TFDSet = Array[0..(FD_MAXFDSETdiv32)-1] of Cardinal`

File descriptor set for `fpSelect` (131) call.

`TFilDes = Array[0..1] of cInt`

Array of file descriptors as used in `fpPipe` (127) call.

`tfpreg = record`
`significand : Array[0..3] of Word;`
`exponent : Word;`
`end`

Record describing floating point register in signal handler.

```
tfpstate = record
  cw : cardinal;
  sw : cardinal;
  tag : cardinal;
  ipoff : cardinal;
  cssel : cardinal;
  dataoff : cardinal;
  datasel : cardinal;
  st : Array[0..7] of tfpreg;
  status : cardinal;
end
```

Record describing floating point unit in signal handler.

TGid = UnixType.TGid

Alias for gid_t (90) type.

TGrpArr = Array[0..0] of TGid

Array of gid_t (90) IDs

timespec = UnixType.timespec

Short time specification type.

timeval = UnixType.timeval

Time specification type.

```
timezone = packed record
  tz_minuteswest : cInt;
  tz_dsttime : cInt;
end
```

time_t = UnixType.time_t

Time span type

TIno = UnixType.TIno

Alias for ino_t (90) type.

TIno64 = Ino64_t

Alias for Ino64_t (90) type.

TMode = UnixType.TMode

Alias for mode_t (90) type.

```

tms = packed record
  tms_utime : clock_t;
  tms_stime : clock_t;
  tms_cutime : clock_t;
  tms_cstime : clock_t;
end

```

Record containing timings for `fpTimes` (144) call.

`TnLink` = `UnixType.TnLink`

Alias for `nlink_t` (90) type.

`TOff` = `UnixType.TOff`

Alias for `off_t` (90) type.

`TOff64` = `Off64_t`

Alias for `Ino64_t` (90) type.

`TPid` = `UnixType.TPid`

Alias for `pid_t` (92) type.

`TSigAction` = `procedure(Sig: LongInt; SigContext: SigContextRec)`

Extended signal handler prototype.

`TSigActionRec` = `SigActionRec`

Alias for `SigActionRec` (94) record type.

`TSigSet` = `SigSet`

Alias for `SigSet` (94) type.

`TSize` = `UnixType.TSize`

Alias for `size_t` (94) type

`TSocklen` = `UnixType.TSocklen`

Alias for `socklen_t` (95) type.

`TsSize` = `UnixType.TsSize`

Alias for `ssize_t` (95) type

`TStat` = `Stat`

Alias for `Stat` (95) type.

```
tstatfs = UnixType.TStatFs
```

```
TTime = UnixType.TTime
```

Alias for TTime (98) type.

```
Ttimespec = UnixType.Ttimespec
```

Alias for TimeSpec (96) type.

```
TTimeVal = UnixType.TTimeVal
```

Alias for timeval (96) type.

```
TTimeZone = timezone
```

```
TTms = tms
```

Alias for Tms (97) record type.

```
TUId = UnixType.TUId
```

Alias for uid_t (98) type.

```
TUtimBuf = UtimBuf
```

Alias for UtimBuf (98) type.

```
TUtsName = UtsName
```

Alias for UtsName (98) type.

```
uid_t = UnixType.uid_t
```

User ID type

```
UtimBuf = record
  actime : time_t;
  modtime : time_t;
end
```

Record used in fpUtime (146) to set file access and modification times.

```
UtsName = record
  Sysname : Array[0..UTSNAME_LENGTH-1] of Char;
  Nodename : Array[0..UTSNAME_NODENAME_LENGTH-1] of Char;
  Release : Array[0..UTSNAME_LENGTH-1] of Char;
  Version : Array[0..UTSNAME_LENGTH-1] of Char;
  Machine : Array[0..UTSNAME_LENGTH-1] of Char;
  Domain : Array[0..UTSNAME_DOMAIN_LENGTH-1] of Char;
end
```

The elements of this record are null-terminated C style strings, you cannot access them directly.

1.4 Procedures and functions

FpAccess

Synopsis: Check file access

Declaration: `function FpAccess(pathname: pChar; aMode: cInt) : cInt`
`function FpAccess(pathname: AnsiString; aMode: cInt) : cInt`

Visibility: default

Description: `FpAccess` tests user's access rights on the specified file. Mode is a mask existing of one or more of the following:

R_OKUser has read rights.

W_OKUser has write rights.

X_OKUser has execute rights.

F_OKFile exists.

The test is done with the real user ID, instead of the effective user ID. If the user has the requested rights, zero is returned. If access is denied, or an error occurred, a nonzero value is returned.

Errors: Extended error information can be retrieved using `fpGetErrno` ([112](#)).

sys_eaccessThe requested access is denied, either to the file or one of the directories in its path.

sys_einvalMode was incorrect.

sys_enoentA directory component in `Path` doesn't exist or is a dangling symbolic link.

sys_enotdirA directory component in `Path` is not a directory.

sys_enomemInsufficient kernel memory.

sys_eloop`Path` has a circular symbolic link.

See also: `FpChown` ([102](#)), `FpChmod` ([101](#))

Listing: `./bunixex/ex26.pp`

Program Example26;

{ Program to demonstrate the Access function. }

Uses BaseUnix;

begin

if `fpAccess ('/etc/passwd', W_OK) = 0` **then**

begin

Writeln ('Better check your system.');

Writeln ('I can write to the /etc/passwd file !');

end;

end.

FpAlarm

Synopsis: Schedule an alarm signal to be delivered

Declaration: `function FpAlarm(seconds: cUInt) : cUInt`

Visibility: default

Description: `FpAlarm` schedules an alarm signal to be delivered to your process in `Seconds` seconds. When `Seconds` seconds have elapsed, the system will send a `SIGALRM` signal to the current process. If `Seconds` is zero, then no new alarm will be set. Whatever the value of `Seconds`, any previous alarm is cancelled.

The function returns the number of seconds till the previously scheduled alarm was due to be delivered, or zero if there was none. A negative value indicates an error.

See also: `fpSigAction` ([134](#)), `fpPause` ([126](#))

Listing: `./bunixex/ex59.pp`

Program `Example59`;

{ Program to demonstrate the Alarm function. }

Uses `BaseUnix`;

Procedure `AlarmHandler(Sig : cint); cdecl`;

begin

Writeln ('Got to alarm handler');
end;

begin

Writeln ('Setting alarm handler');
 `fpSignal(SIGALRM, SignalHandler (@AlarmHandler))`;
 Writeln ('Scheduling Alarm in 10 seconds');
 `fpAlarm(10)`;
 Writeln ('Pausing');
 `fpPause`;
 Writeln ('Pause returned');
end.

FpChdir

Synopsis: Change current working directory.

Declaration: `function FpChdir(path: pChar) : cInt`
 `function FpChdir(path: AnsiString) : cInt`

Visibility: default

Description: `fpChDir` sets the current working directory to `Path`.

It returns zero if the call was succesful, -1 on error.

Errors: Extended error information can be retrieved using `fpGetErrno` ([112](#)).

See also: `fpGetCwd` ([111](#))

FpChmod

Synopsis: Change file permission bits

Declaration: `function FpChmod(path: pChar;Mode: TMode) : cInt`
`function FpChmod(path: AnsiString;Mode: TMode) : cInt`

Visibility: default

Description: `fpChmod` sets the Mode bits of the file in `Path` to `Mode`. Mode can be specified by 'or'-ing the following values:

S_ISUIDSet user ID on execution.

S_ISGIDSet Group ID on execution.

S_ISVTXSet sticky bit.

S_IRUSRRead by owner.

S_IWUSRWrite by owner.

S_IXUSRExecute by owner.

S_IRGRPRead by group.

S_IWGRPWrite by group.

S_IXGRPExecute by group.

S_IROTHRead by others.

S_IWOTHWrite by others.

S_IXOTHExecute by others.

S_IRWXORead, write, execute by others.

S_IRWXGRead, write, execute by groups.

S_IRWXURead, write, execute by user.

If the function is successful, zero is returned. A nonzero return value indicates an error.

Errors: The following error codes are returned:

sys_epermThe effective UID doesn't match the ownership of the file, and is not zero. Owner or group were not specified correctly.

sys_eaccessOne of the directories in `Path` has no search (=execute) permission.

sys_enoentA directory entry in `Path` does not exist or is a symbolic link pointing to a non-existent directory.

sys_enomemInsufficient kernel memory.

sys_erofsThe file is on a read-only filesystem.

sys_eloop`Path` has a reference to a circular symbolic link, i.e. a symbolic link, whose expansion points to itself.

See also: `fpChown` ([102](#)), `fpAccess` ([99](#))

Listing: `./bunixex/ex23.pp`

```

Program Example23;

{ Program to demonstrate the Chmod function . }

Uses BaseUnix, Unix;

Var F : Text;

begin
  { Create a file }
  Assign (f, 'testex21');
  Rewrite (F);
  WriteLn (f, '#!/bin/sh');
  WriteLn (f, 'echo Some text for this file');
  Close (F);
  fpChmod ( 'testex21', &777);
  { File is now executable }
  execl ( './testex21');
end.

```

FpChown

Synopsis: Change owner of file

Declaration: function FpChown(path: pChar;owner: TUid;group: TGid) : cInt
 function FpChown(path: AnsiString;owner: TUid;group: TGid) : cInt

Visibility: default

Description: fpChown sets the User ID and Group ID of the file in Path to Owner,Group.

The function returns zero if the call was succesfull, a nonzero return value indicates an error.

Errors: The following error codes are returned:

sys_epermThe effective UID doesn't match the ownership of the file, and is not zero. Owner or group were not specified correctly.

sys_eaccessOne of the directories in Path has no search (=execute) permission.

sys_enoentA directory entry in Path does not exist or is a symbolic link pointing to a non-existent directory.

sys_enomemInsufficient kernel memory.

sys_erofsThe file is on a read-only filesystem.

sys_eloopPath has a reference to a circular symbolic link, i.e. a symbolic link, whose expansion points to itself.

See also: fpChmod ([101](#)), fpAccess ([99](#))

Listing: ./bunixex/ex24.pp

```

Program Example24;

{ Program to demonstrate the Chown function . }

Uses BaseUnix;

```

```

Var UID : TUid;
      GID : TGid;
      F : Text;

begin

  Writeln ('This will only work if you are root. ');
  Write ('Enter a UID : '); readln(UID);
  Write ('Enter a GID : '); readln(GID);
  Assign (f, 'test.txt');
  Rewrite (f);
  Writeln (f, 'The owner of this file should become : ');
  Writeln (f, 'UID : ',UID);
  Writeln (f, 'GID : ',GID);
  Close (F);
  if fpChown ('test.txt',UID,GID)<>0 then
    if fpgeterrno=ESysEPERM then
      Writeln ('You are not root !')
    else
      Writeln ('Chmod failed with exit code : ',fpgeterrno)
    else
      Writeln ('Changed owner successfully !');
end.

```

FpClose

Synopsis: Close file descriptor

Declaration: function FpClose(fd: cInt) : cInt

Visibility: default

Description: FpClose closes a file with file descriptor Fd. The function returns zero if the file was closed successfully, a nonzero return value indicates an error.

For an example, see FpOpen (124).

Errors: Extended error information can be retrieved using fpGetErrno (112).

See also: FpOpen (124), FpRead (127), FpWrite (148), FpFTruncate (111), FpLSeek (118)

FpClosedir

Synopsis: Close directory file descriptor

Declaration: function FpClosedir(var dirp: Dir) : cInt

Visibility: default

Description: FpCloseDir closes the directory pointed to by dirp. It returns zero if the directory was closed successfully, -1 otherwise.

For an example, see fpOpenDir (125).

Errors: Extended error information can be retrieved using fpGetErrno (112).

See also: FpOpenDir (125), FpReadDir (129)

FpDup

Synopsis: Duplicate a file handle

Declaration: `function FpDup(fildes: cInt) : cInt`
`function FpDup(var oldfile: text;var newfile: text) : cInt`
`function FpDup(var oldfile: file;var newfile: file) : cInt`

Visibility: default

Description: FpDup returns a file descriptor that is a duplicate of the file descriptor fildes.

The second and third forms make NewFile an exact copy of OldFile, after having flushed the buffer of OldFile in case it is a Text file or untyped file. Due to the buffering mechanism of Pascal, these calls do not have the same functionality as the dup call in C. The internal Pascal buffers are not the same after this call, but when the buffers are flushed (e.g. after output), the output is sent to the same file. Doing an lseek will, however, work as in C, i.e. doing a lseek will change the fileposition in both files.

The function returns a negative value in case of an error, a positive value is a file handle, and indicates succes.

Errors: A negative value can be one of the following error codes:

`sys_ebadf`OldFile hasn't been assigned.

`sys_emfile`Maximum number of open files for the process is reached.

See also: fpDup2 ([104](#))

Listing: ./bunixex/ex31.pp

```

program Example31;

  { Program to demonstrate the Dup function. }

uses baseunix;

var f : text;

begin
  if fpdup (output,f)<>0 then
    Writeln ('Dup Failed !');
    writeln ('This is written to stdout. ');
    writeln (f,'This is written to the dup file , and flushed');flush(f);
    writeln
  end.

```

FpDup2

Synopsis: Duplicate one filehandle to another

Declaration: `function FpDup2(fildes: cInt;fildes2: cInt) : cInt`
`function FpDup2(var oldfile: text;var newfile: text) : cInt`
`function FpDup2(var oldfile: file;var newfile: file) : cInt`

Visibility: default

Description: Makes `fildest2` or `NewFile` an exact copy of `fildest` or `OldFile`, after having flushed the buffer of `OldFile` in the case of text or untyped files.

`NewFile` can be an assigned file. If `newfile` was open, it is closed first. Due to the buffering mechanism of Pascal, this has not the same functionality as the `dup2` call in C. The internal Pascal buffers are not the same after this call, but when the buffers are flushed (e.g. after output), the output is sent to the same file. Doing an `lseek` will, however, work as in C, i.e. doing a `lseek` will change the fileposition in both files.

The function returns zero if succesful, a nonzero return value means the call failed.

Errors: In case of error, the following error codes can be reported:

sys_ebadf`OldFile` (or `fildest`) hasn't been assigned.

sys_emfileMaximum number of open files for the process is reached.

See also: `fpDup` ([104](#))

Listing: `./bunixex/ex32.pp`

program Example31;

{ Program to demonstrate the Dup function. }

uses BaseUnix;

var f : text;
i : longint;

begin

Assign (f, 'text.txt');

Rewrite (F);

For i:=1 **to** 10 **do** **writeln** (F, 'Line : ',i);

if `fpdup2` (output,f)<>0 **then**

Writeln ('Dup2 Failed !');

writeln ('This is written to stdout.');

writeln (f, 'This is written to the dup file , and flushed');

flush(f);

writeln;

{ Remove file. Comment this if you want to check flushing. }

`fpUnlink` ('text.txt');

end.

FpExecv

Synopsis: Execute process

Declaration: `function FpExecv(path: pChar;argv: ppChar) : cInt`
`function FpExecv(path: AnsiString;argv: ppchar) : cInt`

Visibility: default

Description: Replaces the currently running program with the program, specified in `path`. It gives the program the options in `argvp`. This is a pointer to an array of pointers to null-terminated strings. The last pointer in this array should be nil. The current environment is passed to the program. On success, `execv` does not return.

Errors: Errors are reported in `LinuxError`:

sys_eaccessFile is not a regular file, or has no execute permission. A component of the path has no search permission.

sys_epermThe file system is mounted \textit{noexec}.

sys_e2bigArgument list too big.

sys_enoexecThe magic number in the file is incorrect.

sys_enoentThe file does not exist.

sys_enomemNot enough memory for kernel.

sys_enotdirA component of the path is not a directory.

sys_eloopThe path contains a circular reference (via symlinks).

See also: `fpExecve` ([106](#)), `fpFork` ([109](#))

Listing: `./bunixex/ex8.pp`

Program Example8;

{ Program to demonstrate the Execv function. }

Uses Unix, strings;

Const Arg0 : PChar = '/bin/lS';
 Arg1 : Pchar = '-l';

Var PP : PPchar;

begin

GetMem (PP, 3* **SizeOf**(Pchar));
 PP[0] := Arg0;
 PP[1] := Arg1;
 PP[3] := **Nil**;
 { Execute '/bin/lS -l', with current environment }
 fpExecv ('/bin/lS', pp);

end.

FpExecve

Synopsis: Execute process using environment

Declaration: `function FpExecve(path: pChar; argv: ppChar; envp: ppChar) : cInt`
 `function FpExecve(path: AnsiString; argv: ppchar; envp: ppchar) : cInt`

Visibility: default

Description: Replaces the currently running program with the program, specified in `path`. It gives the program the options in `argv`, and the environment in `envp`. They are pointers to an array of pointers to null-terminated strings. The last pointer in this array should be `nil`. On success, `execve` does not return.

Errors: Extended error information can be retrieved with `fpGetErrno` ([112](#)), and includes the following:

sys_eaccessFile is not a regular file, or has no execute permission. A component of the path has no search permission.

sys_epermThe file system is mounted \textit{noexec}.

sys_e2bigArgument list too big.
sys_enoexecThe magic number in the file is incorrect.
sys_enoentThe file does not exist.
sys_enomemNot enough memory for kernel.
sys_enotdirA component of the path is not a directory.
sys_eloopThe path contains a circular reference (via symlinks).

See also: `fpExecv` (105), `fpFork` (109)

Listing: `./bunixex/ex7.pp`

Program `Example7`;

{ Program to demonstrate the Execve function. }

Uses `BaseUnix`, `strings`;

Const `Arg0` : `PChar` = `'/bin/lS'`;
 `Arg1` : `Pchar` = `'-l'`;

Var `PP` : `PPchar`;

begin

`GetMem` (`PP`, `3*SizeOf(Pchar)`);
 `PP[0]` := `Arg0`;
 `PP[1]` := `Arg1`;
 `PP[3]` := `Nil`;
 { Execute '/bin/lS -l', with current environment }
 { Env is defined in system.inc }
 `fpExecVe` (`'/bin/lS'`, `pp`, `envp`);

end.

FpExit

Synopsis: Exit the current process

Declaration: `procedure FpExit(Status: cInt)`

Visibility: `default`

Description: `FpExit` exits the currently running process, and report `Status` as the exit status.

Remark: If this call is executed, the normal unit finalization code will not be executed. This may lead to unexpected errors and stray files on your system. It is therefore recommended to use the `Halt` call instead.

Errors: `None`.

See also: `FpFork` (109), `FpExecve` (106)

FpFcntl

Synopsis: File control operations.

Declaration: `function FpFcntl(fildes: cInt;cmd: cInt) : cInt`
`function FpFcntl(fildes: cInt;cmd: cInt;arg: cInt) : cInt`
`function FpFcntl(fildes: cInt;cmd: cInt;var arg: FLock) : cInt`

Visibility: default

Description: Read/set a file's attributes. `Fildes` a valid file descriptor. `Cmd` specifies what to do, and is one of the following:

F_GetFdRead the `close_on_exec` flag. If the low-order bit is 0, then the file will remain open across `execve` calls.

F_GetFIRead the descriptor's flags.

F_GetOwnGet the Process ID of the owner of a socket.

F_SetFdSet the `close_on_exec` flag of `fildes`. (only the least significant bit is used).

F_GetLkReturn the `flock` record that prevents this process from obtaining the lock, or set the `l_type` field of the lock of there is no obstruction. `Arg` is the `flock` record.

F_SetLkSet the lock or clear it (depending on `l_type` in the `flock` structure). if the lock is held by another process, an error occurs.

F_GetLkwSame as for **F_Setlk**, but wait until the lock is released.

F_SetOwnSet the Process or process group that owns a socket.

The function returns 0 if successful, -1 otherwise.

Errors: On error, -1 is returned. Use `fpGetErrno` (112) for extended error information.

sys_ebadfFd has a bad file descriptor.

sys_eagain or sys_eaccessFor \textbf{F_SetLk}, if the lock is held by another process.

fpfdfillset

Synopsis: Set all filedescriptors in the set.

Declaration: `function fpfdfillset(var nset: TFDSets) : cInt`

Visibility: default

Description: `fpfdfillset` sets all filedescriptors in `nset`.

See also: `FpSelect` (131), `FpFD_ZERO` (109), `FpFD_IsSet` (109), `FpFD_Clr` (108), `FpFD_Set` (109)

fpFD_CLR

Synopsis: Clears a filedescriptor in a set

Declaration: `function fpFD_CLR(fdno: cInt;var nset: TFDSets) : cInt`

Visibility: default

Description: `FpFD_Clr` clears file descriptor `fdno` in filedescriptor set `nset`.

For an example, see `FpSelect` (131).

Errors: None.

See also: `FpSelect` (131), `FpFD_ZERO` (109), `FpFD_Set` (109), `FpFD_IsSet` (109)

fpFD_ISSET

Synopsis: Check whether a filedescriptor is set

Declaration: `function fpFD_ISSET(fdno: cInt; const nset: TFDSet) : cInt`

Visibility: default

Description: `FpFD_Set` Checks whether file descriptor `fdNo` in filedescriptor set `fds` is set. It returns zero if the descriptor is not set, 1 if it is set. If the number of the filedescriptor it wrong, -1 is returned.

For an example, see `FpSelect` ([131](#)).

Errors: If an invalid file descriptor number is passed, -1 is returned.

See also: `FpSelect` ([131](#)), `FpFD_ZERO` ([109](#)), `FpFD_Clr` ([108](#)), `FpFD_Set` ([109](#))

fpFD_SET

Synopsis: Set a filedescriptor in a set

Declaration: `function fpFD_SET(fdno: cInt; var nset: TFDSet) : cInt`

Visibility: default

Description: `FpFD_Set` sets file descriptor `fdno` in filedescriptor set `nset`.

For an example, see `FpSelect` ([131](#)).

Errors: None.

See also: `FpSelect` ([131](#)), `FpFD_ZERO` ([109](#)), `FpFD_Clr` ([108](#)), `FpFD_IsSet` ([109](#))

fpFD_ZERO

Synopsis: Clear all file descriptors in set

Declaration: `function fpFD_ZERO(var nset: TFDSet) : cInt`

Visibility: default

Description: `FpFD_ZERO` clears all the filedescriptors in the file descriptor set `nset`.

For an example, see `FpSelect` ([131](#)).

Errors: None.

See also: `FpSelect` ([131](#)), `FpFD_Clr` ([108](#)), `FpFD_Set` ([109](#)), `FpFD_IsSet` ([109](#))

FpFork

Synopsis: Create child process

Declaration: `function FpFork : TPid`

Visibility: default

Description: `FpFork` creates a child process which is a copy of the parent process. `FpFork` returns the process ID in the parent process, and zero in the child's process. (you can get the parent's PID with `fpGetPPid` ([115](#))).

Errors: On error, -1 is returned to the parent, and no child is created.

sys_eagain Not enough memory to create child process.

See also: [fpExecve \(106\)](#), [#rtl.linux.Clone \(597\)](#)

FPFStat

Synopsis: Retrieve file information about a file descriptor.

Declaration: `function FpFStat(fd: cInt; var sb: Stat) : cInt`
`function FPFStat(var F: Text; var Info: Stat) : Boolean`
`function FPFStat(var F: File; var Info: Stat) : Boolean`

Visibility: default

Description: `FpFStat` gets information about the file specified in one of the following:

Fd a valid file descriptor.

F an opened text file or untyped file.

and stores it in `Info`, which is of type `stat` ([95](#)). The function returns zero if the call was successful, a nonzero return value indicates failure.

Errors: Extended error information can be retrieved using `fpGetErrno` ([112](#)).

sys_enoent Path does not exist.

See also: [FpStat \(139\)](#), [FpLStat \(119\)](#)

Listing: `./bunixex/ex28.pp`

```

program example28;

{ Program to demonstrate the FStat function. }

uses BaseUnix;

var f : text;
    i : byte;
    info : stat;

begin
    { Make a file }
    assign (f, 'test.fil');
    rewrite (f);
    for i:=1 to 10 do writeln (f, 'Testline # ', i);
    close (f);
    { Do the call on made file. }
    if fpstat ('test.fil', info) <> 0 then
        begin
            writeln ('Fstat failed. Errno : ', fpgeterrno);
            halt (1);
        end;
    writeln;
    writeln ('Result of fstat on file ''test.fil''.');
    writeln ('Inode    : ', info.st_ino);
    writeln ('Mode     : ', info.st_mode);

```

```

writeln ( 'nlink    : ', info.st_nlink );
writeln ( 'uid      : ', info.st_uid );
writeln ( 'gid      : ', info.st_gid );
writeln ( 'rdev     : ', info.st_rdev );
writeln ( 'Size     : ', info.st_size );
writeln ( 'Blksize  : ', info.st_blksize );
writeln ( 'Blocks   : ', info.st_blocks );
writeln ( 'atime    : ', info.st_atime );
writeln ( 'mtime    : ', info.st_mtime );
writeln ( 'ctime    : ', info.st_ctime );
  { Remove file }
  erase ( f );
end.

```

FpFtruncate

Synopsis: Truncate file on certain size.

Declaration: `function FpFtruncate(fd: cInt; flength: TOff) : cInt`

Visibility: default

Description: `FpFtruncate` sets the length of a file in `fd` on `flength` bytes, where `flength` must be less than or equal to the current length of the file in `fd`.

The function returns zero if the call was successful, a nonzero return value indicates that an error occurred.

Errors: Extended error information can be retrieved using `fpGetErrno` (112).

See also: `FpOpen` (124), `FpClose` (103), `FpRead` (127), `FpWrite` (148), `FpLSeek` (118)

FpGetcwd

Synopsis: Retrieve the current working directory.

Declaration: `function FpGetcwd(path: pChar; siz: TSize) : pChar`
`function FpGetcwd : AnsiString`

Visibility: default

Description: `fpgetCWD` returns the current working directory of the running process. It is returned in `Path`, which points to a memory location of at least `siz` bytes.

If the function is succesful, a pointer to `Path` is returned, or a string with the result. On error `Nil` or an empty string are returned.

Errors: On error `Nil` or an empty string are returned.

See also: `FpGetPID` (114), `FpGetUID` (115)

FpGetegid

Synopsis: Return effective group ID

Declaration: `function FpGetegid : TGid`

Visibility: default

Description: `FpGetegid` returns the effective group ID of the currently running process.

Errors: None.

See also: `FpGetGid` ([113](#)), `FpGetUid` ([115](#)), `FpGetEUid` ([113](#)), `FpGetPid` ([114](#)), `FpGetPPid` ([115](#)), `fpSetUID` ([134](#)), `FpSetGid` ([133](#))

Listing: `./bunixex/ex18.pp`

Program `Example18;`

{ Program to demonstrate the GetGid and GetEGid functions. }

Uses `BaseUnix;`

```
begin
  writeLn ( 'Group Id = ',fpgetgid, ' Effective group Id = ',fpgetegid);
end.
```

FpGetEnv

Synopsis: Return value of environment variable.

Declaration: `function FpGetEnv(name: pChar) : pChar`
`function FpGetEnv(name: String) : pChar`

Visibility: default

Description: `FPGetEnv` returns the value of the environment variable in `Name`. If the variable is not defined, `nil` is returned. The value of the environment variable may be the empty string. A `PChar` is returned to accomodate for strings longer than 255 bytes, `TERMCAP` and `LS_COLORS`, for instance.

Errors: None.

Listing: `./bunixex/ex41.pp`

Program `Example41;`

{ Program to demonstrate the GetEnv function. }

Uses `BaseUnix;`

```
begin
  WriteLn ( 'Path is : ',fpGetenv( 'PATH' ));
end.
```

fpgeterrno

Synopsis: Retrieve extended error information.

Declaration: `function fpgeterrno : LongInt`

Visibility: default

Description: `fpgeterrno` returns extended information on the latest error. It is set by all functions that communicate with the kernel or C library.

Errors: None.

See also: `fpseterrno` ([132](#))

FpGeteuid

Synopsis: Return effective user ID

Declaration: `function FpGeteuid : TUid`

Visibility: `default`

Description: `FpGeteuid` returns the effective user ID of the currently running process.

Errors: None.

See also: `FpGetUid` ([115](#)), `FpGetGid` ([113](#)), `FpGetEGid` ([111](#)), `FpGetPid` ([114](#)), `FpGetPPid` ([115](#)), `fpSetUID` ([134](#)), `FpSetGid` ([133](#))

Listing: `./bunixex/ex17.pp`

Program `Example17;`

{ Program to demonstrate the GetUid and GetEUid functions. }

Uses `BaseUnix;`

begin

`writeln ('User Id = ',fpgetuid , ' Effective user Id = ',fpgeteuid);`
end.

FpGetgid

Synopsis: Return real group ID

Declaration: `function FpGetgid : TGid`

Visibility: `default`

Description: `FpGetgid` returns the real group ID of the currently running process.

Errors: None.

See also: `FpGetEGid` ([111](#)), `FpGetUid` ([115](#)), `FpGetEUid` ([113](#)), `FpGetPid` ([114](#)), `FpGetPPid` ([115](#)), `fpSetUID` ([134](#)), `FpSetGid` ([133](#))

Listing: `./bunixex/ex18.pp`

Program `Example18;`

{ Program to demonstrate the GetGid and GetEGid functions. }

Uses `BaseUnix;`

begin

`writeln ('Group Id = ',fpgetgid , ' Effective group Id = ',fpgetegid);`
end.

FpGetgroups

Synopsis: Get the list of supplementary groups.

Declaration: `function FpGetgroups(gidsetsize: cInt; var grouplist: TGrpArr) : cInt`

Visibility: default

Description: FpGetgroups returns up to gidsetsize groups in GroupList

If the function is successful, then number of groups that were stored is returned. On error, -1 is returned.

Errors: On error, -1 is returned. Extended error information can be retrieved with fpGetErrNo ([112](#))

See also: FpGetpgrp ([114](#)), FpGetGID ([113](#)), FpGetEGID ([111](#))

FpGetpgrp

Synopsis: Get process group ID

Declaration: `function FpGetpgrp : TPid`

Visibility: default

Description: FpGetpgrp returns the process group ID of the current process.

Errors: None.

See also: fpGetPID ([114](#)), fpGetPPID ([115](#)), FpGetGID ([113](#)), FpGetUID ([115](#))

FpGetpid

Synopsis: Return current process ID

Declaration: `function FpGetpid : TPid`

Visibility: default

Description: FpGetpid returns the Process ID of the currently running process.

Errors: None.

See also: FpGetPPid ([115](#))

Listing: ./bunixex/ex16.pp

Program Example16;

{ Program to demonstrate the GetPid , GetPPid function . }

Uses BaseUnix;

begin

WriteLn ('Process Id = ',fpgetpid , ' Parent process Id = ',fpgetppid);

end.

FpGetppid

Synopsis: Return parent process ID

Declaration: `function FpGetppid : TPid`

Visibility: default

Description: `FpGetppid` returns the Process ID of the parent process.

Errors: None.

See also: `FpGetPid` ([114](#))

Listing: `./bunixex/ex16.pp`

Program `Example16;`

{ Program to demonstrate the GetPid, GetPPid function. }

Uses `BaseUnix;`

begin

`WriteLn ('Process Id = ',fpgetpid, ' Parent process Id = ',fpgetppid);`
end.

fpGetPriority

Synopsis: Return process priority

Declaration: `function fpGetPriority(Which: cInt;Who: cInt) : cInt`

Visibility: default

Description: `GetPriority` returns the priority with which a process is running. Which process(es) is determined by the `Which` and `Who` variables. `Which` can be one of the pre-defined `Prio_Process`, `Prio_PGrp`, `Prio_User`, in which case `Who` is the process ID, Process group ID or User ID, respectively.

For an example, see `FpNice` ([123](#)).

Errors: Error information is returned solely by the `FpGetErrno` ([112](#)) function: a priority can be a positive or negative value.

sys_esrchNo process found using `which` and `who`.

sys_einval`Which` was not one of `Prio_Process`, `Prio_Grp` or `Prio_User`.

See also: `FpSetPriority` ([133](#)), `FpNice` ([123](#))

FpGetuid

Synopsis: Return current user ID

Declaration: `function FpGetuid : TUid`

Visibility: default

Description: `FpGetuid` returns the real user ID of the currently running process.

Errors: None.

See also: [FpGetGid \(113\)](#), [FpGetEuid \(113\)](#), [FpGetEGid \(111\)](#), [FpGetPid \(114\)](#), [FpGetPPid \(115\)](#), [fpSetUID \(134\)](#)

Listing: ./bunixex/ex17.pp

Program Example17;

{ Program to demonstrate the GetUid and GetEuid functions. }

Uses BaseUnix;

```
begin
  writeln ( 'User Id = ',fpgetuid , ' Effective user Id = ',fpgeteuid);
end.
```

FpIOCtl

Synopsis: General kernel IOCTL call.

Declaration: function FpIOCtl(Handle: cInt;Ndx: cuLong;Data: Pointer) : cInt

Visibility: default

Description: This is a general interface to the Unix/ linux ioctl call. It performs various operations on the filedescriptor Handle. Ndx describes the operation to perform. Data points to data needed for the Ndx function. The structure of this data is function-dependent, so we don't elaborate on this here. For more information on this, see various manual pages under linux.

Errors: Extended error information can be retrieved using [fpGetErrno \(112\)](#).

Listing: ./bunixex/ex54.pp

Program Example54;

uses BaseUnix,Termio;

{ Program to demonstrate the IOCtl function. }

```
var
  tios : Termios;

begin
  {$ifdef FreeBSD}
    fpIOCtl(1,TIOCGETA,@tios); // these constants are very OS dependant.
                                // see the tcgetattr example for a better way
  {$endif}
  WriteLn( 'Input Flags : $',hexstr(tios.c_iflag,8));
  WriteLn( 'Output Flags : $',hexstr(tios.c_oflag,8));
  WriteLn( 'Line Flags : $',hexstr(tios.c_lflag,8));
  WriteLn( 'Control Flags: $',hexstr(tios.c_cflag,8));
end.
```

FpKill

Synopsis: Send a signal to a process

Declaration: `function FpKill(pid: TPid;sig: cInt) : cInt`

Visibility: default

Description: `fpKill` sends a signal `Sig` to a process or process group. If `Pid>0` then the signal is sent to `Pid`, if it equals `-1`, then the signal is sent to all processes except process 1. If `Pid<-1` then the signal is sent to process group `-Pid`.

The return value is zero, except in case three, where the return value is the number of processes to which the signal was sent.

Errors: Extended error information can be retrieved using `fpGetErrno` (112):

sys_einvalAn invalid signal is sent.

sys_esrchThe `Pid` or process group don't exist.

sys_epermThe effective userid of the current process doesn't math the one of process `Pid`.

See also: `FpSigAction` (134), `FpSignal` (137)

FpLink

Synopsis: Create a hard link to a file

Declaration: `function FpLink(existing: pChar;newone: pChar) : cInt`
`function FpLink(existing: AnsiString;newone: AnsiString) : cInt`

Visibility: default

Description: `fpLink` makes `NewOne` point to the same file als `Existing`. The two files then have the same inode number. This is known as a 'hard' link. The function returns zero if the call was succesfull, and returns a non-zero value if the call failed.

Errors: The following error codes are returned:

sys_exdev`Existing` and `NewOne` are not on the same filesystem.

sys_epermThe filesystem containing `Existing` and `NewOne` doesn't support linking files.

sys_eaccessWrite access for the directory containing `NewOne` is disallowed, or one of the directories in `Existing` or `NewOne` has no search (=execute) permission.

sys_enoentA directory entry in `Existing` or `NewOne` does not exist or is a symbolic link pointing to a non-existent directory.

sys_enotdirA directory entry in `Existing` or `NewOne` is nor a directory.

sys_enomemInsufficient kernel memory.

sys_erofsThe files are on a read-only filesystem.

sys_eexist`NewOne` already exists.

sys_mlink`Existing` has reached maximal link count.

sys_eloop`existing` or `NewOne` has a reference to a circular symbolic link, i.e. a symbolic link, whose expansion points to itself.

sys_enospcThe device containing `NewOne` has no room for another entry.

sys_eperm`Existing` points to `.` or `..` of a directory.

See also: `fpSymLink` (140), `fpUnLink` (145)

Listing: `./bunixex/ex21.pp`

```

Program Example21;

{ Program to demonstrate the Link and UnLink functions. }

Uses BaseUnix;

Var F : Text;
    S : String;
begin
    Assign (F, 'test.txt');
    Rewrite (F);
    WriteIn (F, 'This is written to test.txt');
    Close(f);
    { new.txt and test.txt are now the same file }
    if fpLink ('test.txt', 'new.txt') <> 0 then
        writeln ('Error when linking !');
    { Removing test.txt still leaves new.txt }
    If fpUnlink ('test.txt') <> 0 then
        WriteIn ('Error when unlinking !');
    Assign (f, 'new.txt');
    Reset (F);
    While not EOF(f) do
        begin
            ReadIn(F,S);
            WriteIn ('> ',s);
        end;
    Close (f);
    { Remove new.txt also }
    If not FPUnlink ('new.txt') <> 0 then
        WriteIn ('Error when unlinking !');
end.

```

FpLseek

Synopsis: Set file pointer position.

Declaration: function FpLseek(fd: cInt; offset: TOff; whence: cInt) : TOff

Visibility: default

Description: FpLseek sets the current fileposition of file fd to Offset, starting from Whence, which can be one of the following:

Seek_SetOffset is the absolute position in the file.

Seek_CurOffset is relative to the current position.

Seek_endOffset is relative to the end of the file.

The function returns the new fileposition, or -1 of an error occurred.

For an example, see FpOpen ([124](#)).

Errors: Extended error information can be retrieved using fpGetErrno ([112](#)).

See also: FpOpen ([124](#)), FpWrite ([148](#)), FpClose ([103](#)), FpRead ([127](#)), FpFTruncate ([111](#))

fpLstat

Synopsis: Return information about symbolic link. Do not follow the link

Declaration: `function fpLstat(path: pchar;Info: PStat) : cInt`
`function fpLstat(Filename: ansistring;Info: PStat) : cInt`

Visibility: default

Description: `FpLstat` gets information about the link specified in `Path` (or `FileName`, and stores it in `Info`, which points to a record of type `TStat`. Contrary to `FpFstat` (110), it stores information about the link, not about the file the link points to. The function returns zero if the call was succesful, a nonzero return value indicates failure. failed.

Errors: Extended error information is returned by the `FpGetErrno` (112) function.

`sys_enoent`Path does not exist.

See also: `FpFStat` (110), `#rtl.unix.StatFS` (1269)

Listing: `./unixex/ex29.pp`

```

program example29;

{ Program to demonstrate the LStat function. }

uses BaseUnix, Unix;

var f : text;
    i : byte;
    info : stat;

begin
  { Make a file }
  assign (f, 'test.fil ');
  rewrite (f);
  for i:=1 to 10 do writeln (f, 'Testline # ', i);
  close (f);
  { Do the call on made file. }
  if fpstat ('test.fil ', info)<>0 then
    begin
      writeln('Fstat failed. Errno : ', fpgeterrno);
      halt (1);
    end;
  writeln;
  writeln ('Result of stat on file ''test.fil''.');
  writeln ('Inode   : ', info.st_ino);
  writeln ('Mode    : ', info.st_mode);
  writeln ('nlink   : ', info.st_nlink);
  writeln ('uid     : ', info.st_uid);
  writeln ('gid     : ', info.st_gid);
  writeln ('rdev    : ', info.st_rdev);
  writeln ('Size    : ', info.st_size);
  writeln ('Blksize  : ', info.st_blksize);
  writeln ('Blocks  : ', info.st_blocks);
  writeln ('atime   : ', info.st_atime);
  writeln ('mtime   : ', info.st_mtime);
  writeln ('ctime   : ', info.st_ctime);

  if fpSymLink ('test.fil ', 'test.lnk')<>0 then

```

```

    writeln ( 'Link failed ! Errno : ',fpgeterrno);

    if fplstat ( 'test.lnk',@info)<>0 then
    begin
        writeln('LStat failed. Errno : ',fpgeterrno);
        halt (1);
    end;
    writeln;
    writeln ( 'Result of fstat on file ''test.lnk''.' );
    writeln ( 'Inode   : ',info.st_ino);
    writeln ( 'Mode    : ',info.st_mode);
    writeln ( 'nlink   : ',info.st_nlink);
    writeln ( 'uid     : ',info.st_uid);
    writeln ( 'gid     : ',info.st_gid);
    writeln ( 'rdev    : ',info.st_rdev);
    writeln ( 'Size    : ',info.st_size);
    writeln ( 'Blksize  : ',info.st_blksize);
    writeln ( 'Blocks   : ',info.st_blocks);
    writeln ( 'atime   : ',info.st_atime);
    writeln ( 'mtime   : ',info.st_mtime);
    writeln ( 'ctime   : ',info.st_ctime);
    { Remove file and link }
    erase (f);
    fpunlink ( 'test.lnk' );
end.

```

FpMkdir

Synopsis: Create a new directory

Declaration: `function FpMkdir(path: pChar;Mode: TMode) : cInt`
`function FpMkdir(path: AnsiString;Mode: TMode) : cInt`

Visibility: default

Description: `FpMkDir` creates a new directory `Path`, and sets the new directory's mode to `Mode`. `Path` can be an absolute path or a relative path. Note that only the last element of the directory will be created, higher level directories must already exist, and must be writeable by the current user.

On succes, 0 is returned. if the function fails, -1 is returned.

Errors: Extended error information can be retrieved using `fpGetErrno` ([112](#)).

See also: `fpGetCWD` ([111](#)), `fpChDir` ([100](#))

FpMkfifo

Synopsis: Create FIFO (named pipe) in file system

Declaration: `function FpMkfifo(path: pChar;Mode: TMode) : cInt`
`function FpMkfifo(path: AnsiString;Mode: TMode) : cInt`

Visibility: default

Description: `fpMkFifo` creates a named pipe in the filesystem, with name `Path` and mode `Mode`.

The function returns zero if the command was succesful, and nonzero if it failed.

Errors: The error codes include:

sys_emfile Too many file descriptors for this process.

sys_enfile The system file table is full.

Fpmmmap

Synopsis: Create memory map of a file

Declaration: `function Fpmmmap(start: pointer; len: size_t; prot: cInt; flags: cInt; fd: cInt; offst: off_t) : pointer`

Visibility: default

Description: FpMMMap maps or unmaps files or devices into memory. The different arguments determine what and how the file is mapped:

adr Address where to mmap the device. This address is a hint, and may not be followed.

len Size (in bytes) of area to be mapped.

prot Protection of mapped memory. This is a OR-ed combination of the following constants:

PROT_EXEC The memory can be executed.

PROT_READ The memory can be read.

PROT_WRITE The memory can be written.

PROT_NONE The memory can not be accessed.

flags Contains some options for the mmap call. It is an OR-ed combination of the following constants:

MAP_FIXED Do not map at another address than the given address. If the address cannot be used, MMMap will fail.

MAP_SHARED Share this map with other processes that map this object.

MAP_PRIVATE Create a private map with copy-on-write semantics.

MAP_ANONYMOUS fd does not have to be a file descriptor.

One of the options MAP_SHARED and MAP_PRIVATE must be present, but not both at the same time.

fd File descriptor from which to map.

off Offset to be used in file descriptor fd.

The function returns a pointer to the mapped memory, or a -1 in case of an error.

Errors: On error, -1 is returned and extended error information is returned by the FpGetErrno (112) function.

Sys_EBADF fd is not a valid file descriptor and MAP_ANONYMOUS was not specified.

Sys_EACCES MAP_PRIVATE was specified, but fd is not open for reading. Or MAP_SHARED was asked and PROT_WRITE is set, fd is not open for writing

Sys_EINVAL One of the record fields Start, length or offset is invalid.

Sys_ETXTBUSY MAP_DENYWRITE was set but the object specified by fd is open for writing.

Sys_EAGAIN fd is locked, or too much memory is locked.

Sys_ENOMEM Not enough memory for this operation.

See also: FpMUnMap (122)

Listing: ./unixex/ex66.pp

Program Example66;

{ Program to demonstrate the MMap function. }

Uses BaseUnix, Unix;

Var S : **String**;
 fd : **cint**;
 Len : **longint**;
 // args : *tmmmapargs*;
 P : **PChar**;

begin

```

s:= 'This is the string';
Len:=Length(S);
fd:=fpOpen('testfile.txt',O_wrOnly or o_creat);
If fd=-1 then
  Halt(1);
If fpWrite(fd,S[1],Len)=-1 then
  Halt(2);
fpClose(fd);
fd:=fpOpen('testfile.txt',O_rdOnly);
if fd=-1 then
  Halt(3);
P:=Pchar(fpmmap(nil,len+1,PROT_READ or PROT_WRITE,MAP_PRIVATE,fd,0));

If longint(P)=-1 then
  Halt(4);
WriteIn('Read in memory :',P);
fpclose(fd);
if fpMUnMap(P,Len)<>0 Then
  Halt(fpgeterrno);

```

end.

Fpmunmap

Synopsis: Unmap previously mapped memory block

Declaration: `function Fpmunmap(start: pointer;len: size_t) : cInt`

Visibility: default

Description: `FpMUnMap` unmaps the memory block of size `Len`, pointed to by `Adr`, which was previously allocated with `FpMMap` (121).

The function returns `True` if successful, `False` otherwise.

For an example, see `FpMMap` (121).

Errors: In case of error the function returns a nonzero value, extended error information is returned by the `FpGetErrno` (112) function. See `FpMMap` (121) for possible error values.

See also: `FpMMap` (121)

FpNanoSleep

Synopsis: Suspend process for a short time

Declaration: `function FpNanoSleep(req: ptimespec;rem: ptimespec) : cInt`

Visibility: default

Description: `FpNanoSleep` suspends the process till a time period as specified in `req` has passed. Then the function returns. If the call was interrupted (e.g. by some signal) then the function may return earlier, and `rem` will contain the remaining time till the end of the intended period. In this case the return value will be -1, and `LinuxError` will be set to `EINTR`.

If the function returns without error, the return value is zero.

Errors: If an error occurred or the call was interrupted, -1 is returned. Extended error information can be retrieved using `fpGetErrno` (112).

See also: `FpPause` (126), `FpAlarm` (100)

Listing: `./bunixex/ex72.pp`

```

program example72;

{ Program to demonstrate the NanoSleep function. }

uses BaseUnix;

Var
  Req,Rem : TimeSpec;
  Res : Longint;

begin
  With Req do
    begin
      tv_sec:=10;
      tv_nsec:=100;
    end;
  Write( 'NanoSleep returned : ');
  Flush(Output);
  Res:=(fpNanoSleep(@Req,@rem));
  WriteLn(res);
  If (res<>0) then
    With rem do
      begin
        WriteLn( 'Remaining seconds      : ',tv_sec);
        WriteLn( 'Remaining nanoseconds : ',tv_nsec);
      end;
  end.
```

fpNice

Synopsis: Set process priority

Declaration: `function fpNice(N: cInt) : cInt`

Visibility: default

Description: `Nice` adds `-N` to the priority of the running process. The lower the priority numerically, the less the process is favored. Only the superuser can specify a negative `N`, i.e. increase the rate at which the process is run.

If the function is succesful, zero is returned. On error, a nonzero value is returned.

Errors: Extended error information is returned by the `FpGetErrno` (112) function.

sys_eperm A non-superuser tried to specify a negative N, i.e. do a priority increase.

See also: `FpGetPriority` (115), `FpSetPriority` (133)

Listing: ./unixex/ex15.pp

Program Example15;

{ Program to demonstrate the Nice and Get/SetPriority functions. }

Uses BaseUnix, Unix;

```
begin
  writeln ( 'Setting priority to 5' );
  fpsetpriority ( prio_process, fpgetpid, 5 );
  writeln ( 'New priority = ', fpgetpriority ( prio_process, fpgetpid ) );
  writeln ( 'Doing nice 10' );
  fpnice ( 10 );
  writeln ( 'New Priority = ', fpgetpriority ( prio_process, fpgetpid ) );
end.
```

FpOpen

Synopsis: Open file and return file descriptor

Declaration: `function FpOpen(path: pChar; flags: cInt; Mode: TMode) : cInt`
`function FpOpen(path: pChar; flags: cInt) : cInt`
`function FpOpen(path: AnsiString; flags: cInt) : cInt`
`function FpOpen(path: AnsiString; flags: cInt; Mode: TMode) : cInt`
`function FpOpen(path: String; flags: cInt) : cInt`
`function FpOpen(path: String; flags: cInt; Mode: TMode) : cInt`

Visibility: default

Description: `FpOpen` opens a file in `Path` with flags `flags` and mode `Mode` One of the following:

O_RdOnlyFile is opened Read-only

O_WrOnlyFile is opened Write-only

O_RdWrFile is opened Read-Write

The flags may be OR-ed with one of the following constants:

O_CreatFile is created if it doesn't exist.

O_Excl If the file is opened with `O_Creat` and it already exists, the call will fail.

O_NoCtty If the file is a terminal device, it will NOT become the process' controlling terminal.

O_Trunc If the file exists, it will be truncated.

O_Append the file is opened in append mode. *Before each write*, the file pointer is positioned at the end of the file.

O_NonBlock The file is opened in non-blocking mode. No operation on the file descriptor will cause the calling process to wait till.

O_NDelay Idem as `O_NonBlock`

O_SyncThe file is opened for synchronous IO. Any write operation on the file will not return until the data is physically written to disk.

O_NoFollowif the file is a symbolic link, the open fails. (linux 2.1.126 and higher only)

O_Directoryif the file is not a directory, the open fails. (linux 2.1.126 and higher only)

Path can be of type PChar or String. The optional mode argument specifies the permissions to set when opening the file. This is modified by the umask setting. The real permissions are Mode and not umask. The return value of the function is the filedescriptor, or a negative value if there was an error.

Errors: Extended error information can be retrieved using fpGetErrno (112).

See also: FpClose (103), FpRead (127), FpWrite (148), FpFTruncate (111), FpLSeek (118)

Listing: ./bunixex/ex19.pp

Program Example19;

{ Program to demonstrate the fdOpen, fdwrite and fdClose functions. }

Uses BaseUnix;

Const Line : **String**[80] = 'This is easy writing !';

Var FD : CInt;

begin

FD:=fpOpen ('Test.dat',O_WrOnly or O_Creat);

if FD>0 then

begin

if length(Line)<>fpwrite (FD,Line[1],Length(Line)) then

Writeln ('Error when writing to file !');

fpClose(FD);

end;

end.

FpOpendir

Synopsis: Open a directory for reading

Declaration: function FpOpendir(dirname: pChar) : pDir
function FpOpendir(dirname: AnsiString) : pDir
function FpOpendir(dirname: shortString) : pDir

Visibility: default

Description: FpOpenDir opens the directory DirName, and returns a pdir pointer to a Dir (90) record, which can be used to read the directory structure. If the directory cannot be opened, nil is returned.

Errors: Extended error information can be retrieved using fpGetErrno (112).

See also: FpCloseDir (103), FpReadDir (129)

Listing: ./bunixex/ex35.pp

Program Example35;

*{ Program to demonstrate the
OpenDir, ReadDir, SeekDir and TellDir functions. }*

Uses BaseUnix;

Var TheDir : PDir;
ADirent : PDirent;
Entry : Longint;

begin

TheDir:=fpOpenDir('./. ');

Repeat

// Entry:=fpTellDir (TheDir);

ADirent:=fpReadDir (TheDir^);

If ADirent<>Nil **then**

With ADirent^ **do**

begin

Writeln ('Entry No : ',Entry);

Writeln ('Inode : ',d_fileno);

// *Writeln* ('Offset : ',d_off);

Writeln ('Reclen : ',d_reclen);

Writeln ('Name : ',pchar(@d_name[0]));

end;

Until ADirent=Nil;

Repeat

Write ('Entry No. you would like to see again (-1 to stop): ');

ReadLn (Entry);

If Entry<>-1 **then**

begin

// fpSeekDir (TheDir, Entry);

// not implemented for various platforms

ADirent:=fpReadDir (TheDir^);

If ADirent<>Nil **then**

With ADirent^ **do**

begin

Writeln ('Entry No : ',Entry);

Writeln ('Inode : ',d_fileno);

// *Writeln* ('Offset : ',d_off);

Writeln ('Reclen : ',d_reclen);

Writeln ('Name : ',pchar(@d_name[0]));

end;

end;

Until Entry=-1;

fpCloseDir (TheDir^);

end.

FpPause

Synopsis: Wait for a signal to arrive

Declaration: function FpPause : cInt

Visibility: default

Description: FpPause puts the process to sleep and waits until the application receives a signal. If a signal handler is installed for the received sigal, the handler will be called and after that pause will return

control to the process.

For an example, see `fpAlarm` ([100](#)).

FpPipe

Synopsis: Create a set of pipe file handlers

Declaration: `function FpPipe(var fildes: TFilDes) : cInt`

Visibility: default

Description: `FpPipe` creates a pipe, i.e. two file objects, one for input, one for output. The filehandles are returned in the array `fildes`. The input handle is in the 0-th element of the array, the output handle is in the 1-st element.

The function returns zero if everything went succesfully, a nonzero return value indicates an error.

Errors: In case the function fails, the following return values are possible:

sys_enfile Too many file descriptors for this process.

sys_enfile The system file table is full.

See also: `#rtl.unix.POpen` ([1265](#)), `fpMkFifo` ([120](#))

Listing: `./bunixex/ex36.pp`

Program Example36;

{ Program to demonstrate the AssignPipe function. }

Uses BaseUnix, Unix;

Var pipi, pipo : Text;
s : String;

```
begin
  Writeln ('Assigning Pipes. ');
  If assignpipe(pipi, pipo) <> 0 then
    Writeln ('Error assigning pipes !', fpgeterrno);
  Writeln ('Writing to pipe, and flushing. ');
  Writeln (pipo, 'This is a textstring'); close(pipo);
  Writeln ('Reading from pipe. ');
  While not eof(pipi) do
    begin
      Readln (pipi, s);
      Writeln ('Read from pipe : ', s);
    end;
  close (pipi);
  writeln ('Closed pipes. ');
  writeln
end.
```

FpRead

Synopsis: Read data from file descriptor

Declaration: `function FpRead(fd: cInt;buf: pChar;nbytes: TSize) : TsSize`
`function FpRead(fd: cInt;var buf;nbytes: TSize) : TsSize`

Visibility: default

Description: `FpRead` reads at most `nbytes` bytes from the file descriptor `fd`, and stores them in `buf`.

The function returns the number of bytes actually read, or -1 if an error occurred. No checking on the length of `buf` is done.

Errors: Extended error information can be retrieved using `fpGetErrno` (112).

See also: `FpOpen` (124), `FpClose` (103), `FpWrite` (148), `FpFTruncate` (111), `FpLSeek` (118)

Listing: `./bunixex/ex20.pp`

Program `Example20`;

{ Program to demonstrate the fdRead and fdTruncate functions. }

Uses `BaseUnix`;

Const `Data : string[10] = '1234567890'`;

Var `FD : cint;`
`l : longint;`

begin

`FD:=fpOpen('test.dat',o_wronly or o_creat,&666);`

`if fd>0 then`

`begin`

{ Fill file with data }

`for l:=1 to 10 do`

`if fpWrite (FD,Data[l],10)<>10 then`

`begin`

`writeln ('Error when writing !');`

`halt(1);`

`end;`

`fpClose(FD);`

`FD:=fpOpen('test.dat',o_rdonly);`

{ Read data again }

`If FD>0 then`

`begin`

`For l:=1 to 5 do`

`if fpRead (FD,Data[l],10)<>10 then`

`begin`

`Writeln ('Error when Reading !');`

`Halt(2);`

`end;`

`fpClose(FD);`

{ Truncating file at 60 bytes }

{ For truncating , file must be open or write }

`FD:=fpOpen('test.dat',o_wronly,&666);`

`if FD>0 then`

`begin`

`if fpfTruncate (FD,60)<>0 then`

`Writeln('Error when truncating !');`

`fpClose (FD);`

`end;`

`end;`

```

    end;
end.

```

FpReaddir

Synopsis: Read entry from directory

Declaration: `function FpReaddir(var dirp: Dir) : pDirent`

Visibility: default

Description: `FpReadDir` reads the next entry in the directory pointed to by `dirp`. It returns a `pdirent` pointer to a `dirent` (90) record describing the entry. If the next entry can't be read, `Nil` is returned.

For an example, see `FpOpenDir` (125).

Errors: Extended error information can be retrieved using `fpGetErrno` (112).

See also: `FpCloseDir` (103), `FpOpenDir` (125)

fpReadLink

Synopsis: Read destination of symbolic link

Declaration: `function fpReadLink(name: pchar; linkname: pchar; maxlen: size_t) : cInt`
`function fpReadLink(Name: ansistring) : ansistring`

Visibility: default

Description: `FpReadLink` returns the file the symbolic link name is pointing to. The first form of this function accepts a buffer `linkname` of length `maxlen` where the filename will be stored. It returns the actual number of characters stored in the buffer.

The second form of the function returns simply the name of the file.

Errors: On error, the first form of the function returns -1; the second one returns an empty string. Extended error information is returned by the `FpGetErrno` (112) function.

SYS_ENOTDIRA part of the path in `Name` is not a directory.

SYS_EINVAL`maxlen` is not positive, or the file is not a symbolic link.

SYS_ENAMETOOLONGA pathname, or a component of a pathname, was too long.

SYS_ENOENTthe link name does not exist.

SYS_EACCESNo permission to search a directory in the path

SYS_ELOOPToo many symbolic links were encountered in translating the pathname.

SYS_EIOAn I/O error occurred while reading from the file system.

SYS_EFAULTThe buffer is not part of the process's memory space.

SYS_ENOMEMNot enough kernel memory was available.

See also: `FpSymLink` (140)

Listing: `./unixex/ex62.pp`

Program Example62;

{ Program to demonstrate the ReadLink function. }

Uses BaseUnix, Unix;

Var F : Text;
 S : **String**;

begin
 Assign (F, 'test.txt');
 Rewrite (F);
 Writeln (F, 'This is written to test.txt');
 Close(f);
 { new.txt and test.txt are now the same file }
 if fpSymLink ('test.txt', 'new.txt') <> 0 **then**
 writeln ('Error when symlinking !');
 S:=fpReadLink('new.txt');
 If S='' **then**
 Writeln ('Error reading link !')
 Else
 Writeln ('Link points to : ',S);
 { Now remove links }
 If fpUnlink ('new.txt') <> 0 **then**
 Writeln ('Error when unlinking !');
 If fpUnlink ('test.txt') <> 0 **then**
 Writeln ('Error when unlinking !');
end.

FpRename

Synopsis: Rename file

Declaration: function FpRename(old: pChar; newpath: pChar) : cInt
 function FpRename(old: AnsiString; newpath: AnsiString) : cInt

Visibility: default

Description: FpRename renames the file Old to NewPath. NewPath can be in a different directory than Old, but it cannot be on another partition (device). Any existing file on the new location will be replaced.

If the operation fails, then the Old file will be preserved.

The function returns zero on succes, a nonzero value indicates failure.

Errors: Extended error information can be retrieved using fpGetErrno ([112](#)).

sys_eisdirNewPath exists and is a directory, but Old is not a directory.

sys_exdevNewPath and Old are on different devices.

sys_enotempty or **sys_eexist**NewPath is an existing, non-empty directory.

sys_ebusyOld or NewPath is a directory and is in use by another process.

sys_einvalNewPath is part of Old.

sys_emlinkOldPath or NewPath already have the maximum amount of links pointing to them.

sys_enotdirpart of Old or NewPath is not directory.

sys_efaultFor the pchar case: One of the pointers points to an invalid address.

sys_eaccess access is denied when attempting to move the file.

sys_enametoolong Either Old or NewPath is too long.

sys_enoent a directory component in Old or NewPath didn't exist.

sys_enomem not enough kernel memory.

sys_erofs NewPath or Old is on a read-only file system.

sys_eloop too many symbolic links were encountered trying to expand Old or NewPath

sys_enosp the filesystem has no room for the new directory entry.

See also: [FpUnLink \(145\)](#)

FpRmdir

Synopsis: Remove a directory.

Declaration: `function FpRmdir(path: pChar) : cInt`
`function FpRmdir(path: AnsiString) : cInt`

Visibility: default

Description: `FpRmdir` removes the directory Path from the system. The directory must be empty for this call to succeed, and the user must have the necessary permissions in the parent directory. Only the last component of the directory is removed, i.e. higher-lying directories are not removed.

On success, zero is returned. A nonzero return value indicates failure.

Errors: Extended error information can be retrieved using [fpGetErrno \(112\)](#).

fpSelect

Synopsis: Wait for events on file descriptors

Declaration: `function FPSelect(N: cInt; readfds: pFDSet; writefds: pFDSet;`
`exceptfds: pFDSet; Timeout: ptimeval) : cInt`
`function fpSelect(N: cInt; readfds: pFDSet; writefds: pFDSet;`
`exceptfds: pFDSet; Timeout: cInt) : cInt`
`function fpSelect(var T: Text; Timeout: ptimeval) : cInt`
`function fpSelect(var T: Text; Timeout: time_t) : cInt`

Visibility: default

Description: `fpSelect` checks one of the file descriptors in the `FDSet`s to see if its status changed.

`readfds`, `writefds` and `exceptfds` are pointers to arrays of 256 bits. If you want a file descriptor to be checked, you set the corresponding element in the array to 1. The other elements in the array must be set to zero. Three arrays are passed : The entries in `readfds` are checked to see if characters become available for reading. The entries in `writefds` are checked to see if it is OK to write to them, while entries in `exceptfds` are checked to see if an exception occurred on them.

You can use the functions [fpFD_ZERO \(109\)](#), [fpFD_Clr \(108\)](#), [fpFD_Set \(109\)](#) or [fpFD_IsSet \(109\)](#) to manipulate the individual elements of a set.

The pointers can be `Nil`.

`N` is the largest index of a nonzero entry plus 1. (= the largest file-descriptor + 1).

`Timeout` can be used to set a time limit. If `Timeout` can be two types :

1. `Timeout` is of type `ptimeval` and contains a zero time, the call returns immediately. If `Timeout` is `Nil`, the kernel will wait forever, or until a status changed.
2. `Timeout` is of type `cint`. If it is -1, this has the same effect as a `Timeout` of type `PTime` which is `Nil`. Otherwise, `Timeout` contains a time in milliseconds.

When the `Timeout` is reached, or one of the file descriptors has changed, the `Select` call returns. On return, it will have modified the entries in the array which have actually changed, and it returns the number of entries that have been changed. If the timeout was reached, and no descriptor changed, zero is returned; The arrays of indexes are undefined after that. On error, -1 is returned.

The variant with the text file will execute the `FpSelect` call on the file descriptor associated with the text file `T`

Errors: On error, the function returns -1. Extended error information can be retrieved using `fpGetErrno` (112).

SYS_EBADF An invalid descriptor was specified in one of the sets.

SYS_EINTR A non blocked signal was caught.

SYS_EINVAL `n` is negative or too big.

SYS_ENOMEM `Select` was unable to allocate memory for its internal tables.

See also: `fpFD_ZERO` (109), `fpFD_Clr` (108), `fpFD_Set` (109), `fpFD_IsSet` (109)

Listing: `./bunixex/ex33.pp`

Program `Example33`;

{ Program to demonstrate the Select function. }

Uses `BaseUnix`;

Var `FDS` : `Tfdset`;

begin

```

    fpfd_zero(FDS);
    fpfd_set(0,FDS);
    Writeln ( 'Press the <ENTER> to continue the program.' );
    { Wait until File descriptor 0 (=Input) changes }
    fpSelect (1,@FDS,nil , nil , nil );
    { Get rid of <ENTER> in buffer }
    readln;
    Writeln ( 'Press <ENTER> key in less than 2 seconds...' );
    Fpfd_zero(FDS);
    FpFd_set (0 ,FDS);
    if fpSelect (1 ,@FDS, nil , nil ,2000)>0 then
        Writeln ( 'Thank you ! ' )
        { FD_ISSET(0,FDS) would be true here. }
    else
        Writeln ( 'Too late ! ' );
end.
```

fpseterrno

Synopsis: Set extended error information.

Declaration: `procedure fpseterrno(err: LongInt)`

Visibility: default

Description: `fpseterrno` sets the extended information on the latest error. It is called by all functions that communicate with the kernel or C library.

Unless a direct kernel call is performed, there should never be any need to call this function.

Errors:

See also: `fpgeterrno` ([112](#))

FpSetgid

Synopsis: Set the current group ID

Declaration: `function FpSetgid(gid: TGid) : cInt`

Visibility: default

Description: `fpSetUID` sets the group ID of the current process. This call will only work if it is executed as root, or the program is `setgid` root.

On success, zero is returned, on error -1 is returned.

Errors: Extended error information can be retrieved with `fpGetErrNo` ([112](#)).

See also: `FpSetUid` ([134](#)), `FpGetGid` ([113](#)), `FpGetUid` ([115](#)), `FpGetEUid` ([113](#)), `FpGetEGid` ([111](#)), `FpGetPid` ([114](#)), `FpGetPPid` ([115](#))

fpSetPriority

Synopsis: Set process priority

Declaration: `function fpSetPriority(Which: cInt;Who: cInt;What: cInt) : cInt`

Visibility: default

Description: `fpSetPriority` sets the priority with which a process is running. Which process(es) is determined by the `Which` and `Who` variables. Which can be one of the pre-defined constants:

Prio_ProcessWho is interpreted as process ID

Prio_PGrpWho is interpreted as process group ID

Prio_UserWho is interpreted as user ID

`Prio` is a value in the range -20 to 20.

For an example, see `FpNice` ([123](#)).

The function returns zero on success, -1 on failure

Errors: Extended error information is returned by the `FpGetErrno` ([112](#)) function.

sys_esrchNo process found using which and who.

sys_einvalWhich was not one of `Prio_Process`, `Prio_Grp` or `Prio_User`.

sys_epermA process was found, but neither its effective or real user ID match the effective user ID of the caller.

sys_eaccessA non-superuser tried to a priority increase.

See also: `FpGetPriority` ([115](#)), `FpNice` ([123](#))

FpSetsid

Synopsis: Create a new session.

Declaration: `function FpSetsid : TPid`

Visibility: default

Description: `FpSetsid` creates a new session (process group). It returns the new process group id (as returned by `FpGetpgrp` (114)). This call will fail if the current process is already the process group leader.

Errors: On error, -1 is returned. Extended error information can be retrieved with `fpGetErrNo` (112)

fpsettimeofday

Declaration: `function fpsettimeofday(tp: ptimeval; tzp: ptimezone) : cInt`

Visibility: default

FpSetuid

Synopsis: Set the current user ID

Declaration: `function FpSetuid(uid: TUid) : cInt`

Visibility: default

Description: `fpSetUID` sets the user ID of the current process. This call will only work if it is executed as root, or the program is `setuid` root.

On success, zero is returned, on error -1 is returned.

Errors: Extended error information can be retrieved with `fpGetErrNo` (112).

See also: `FpGetGid` (113), `FpGetUid` (115), `FpGetEUid` (113), `FpGetEGid` (111), `FpGetPid` (114), `FpGetPPid` (115), `FpSetGid` (133)

FPSigaction

Synopsis: Install signal handler

Declaration: `function FPSigaction(sig: cInt; act: PSigActionRec; oact: PSigActionRec) : cInt`

Visibility: default

Description: `FPSigaction` changes the action to take upon receipt of a signal. `Act` and `Oact` are pointers to a `SigActionRec` (94) record. `Sig` specifies the signal, and can be any signal except **SIGKILL** or **SIGSTOP**.

If `Act` is non-nil, then the new action for signal `Sig` is taken from it. If `Oact` is non-nil, the old action is stored there. `Sa_Handler` may be `SIG_DFL` for the default action or `SIG_IGN` to ignore the signal. `Sa_Mask` Specifies which signals should be ignored during the execution of the signal handler. `Sa_Flags` Specifies a series of flags which modify the behaviour of the signal handler. You can 'or' none or more of the following :

SA_NOCLDSTOP If `sig` is **SIGCHLD** do not receive notification when child processes stop.

SA_ONESHOT or **SA_RESETHAND** Restore the signal action to the default state once the signal handler has been called.

SA_RESTARTFor compatibility with BSD signals.

SA_NOMASK or SA_NODEFERDo not prevent the signal from being received from within its own signal handler.

Errors: Extended error information can be retrieved using `fpGetErrno` (112).

sys_einvalan invalid signal was specified, or it was **SIGKILL** or **SIGSTOP**.

sys_efault`Act , OldAct` point outside this process address space

sys_eintrSystem call was interrupted.

See also: `FpSigProcMask` (138), `FpSigPending` (138), `FpSigSuspend` (138), `FpKill` (116)

Listing: `./bunixex/ex57.pp`

Program `example57`;

```
{ Program to demonstrate the SigAction function. }

{
do a kill -USR1 pid from another terminal to see what happens.
replace pid with the real pid of this program.
You can get this pid by running 'ps'.
}

uses BaseUnix;

Var
    oa, na : PSigActionRec;

Procedure DoSig(sig : cint); cdecl;

begin
    writeln( 'Receiving signal: ', sig );
end;

begin
    new(na);
    new(oa);
    na^.sa_Handler:=TSigaction(@DoSig);
    fillchar(na^.Sa_Mask, sizeof(na^.sa_mask), #0);
    na^.Sa_Flags:=0;
    { $ifdef Linux }                // Linux specific
    na^.Sa_Restorer:=Nil;
    { $endif }
    if fpSigAction(SigUsr1, na, oa) <> 0 then
        begin
            writeln( 'Error: ', fpgeterrno, '. ' );
            halt(1);
        end;
    Writeln( 'Send USR1 signal or press <ENTER> to exit' );
    readln;
end.
```

FpSigAddSet

Synopsis: Set a signal in a signal set.

Declaration: `function FpSigAddSet(var nset: TSigSet; signo: cInt) : cInt`

Visibility: default

Description: `FpSigAddSet` adds signal `Signo` to the signal set `nset`. The function returns 0 on success.

Errors: If an invalid signal number is given, -1 is returned.

See also: `FpSigEmptySet` ([136](#)), `FpSigFillSet` ([136](#)), `FpSigDelSet` ([136](#)), `FpSigIsMember` ([137](#))

FpSigDelSet

Synopsis: Remove a signal from a signal set.

Declaration: `function FpSigDelSet(var nset: TSigSet; signo: cInt) : cInt`

Visibility: default

Description: `FpSigDelSet` removes signal `Signo` to the signal set `nset`. The function returns 0 on success.

Errors: If an invalid signal number is given, -1 is returned.

See also: `FpSigEmptySet` ([136](#)), `FpSigFillSet` ([136](#)), `FpSigAddSet` ([135](#)), `FpSigIsMember` ([137](#))

FpsigEmptySet

Synopsis: Clear all signals from signal set.

Declaration: `function FpsigEmptySet(var nset: TSigSet) : cInt`

Visibility: default

Description: `FpSigEmptySet` clears all signals from the signal set `nset`.

Errors: None. This function always returns zero.

See also: `FpSigFillSet` ([136](#)), `FpSigAddSet` ([135](#)), `FpSigDelSet` ([136](#)), `FpSigIsMember` ([137](#))

FpSigFillSet

Synopsis: Set all signals in signal set.

Declaration: `function FpSigFillSet(var nset: TSigSet) : cInt`

Visibility: default

Description: `FpSigFillSet` sets all signals in the signal set `nset`.

Errors: None. This function always returns zero.

See also: `FpSigEmptySet` ([136](#)), `FpSigAddSet` ([135](#)), `FpSigDelSet` ([136](#)), `FpSigIsMember` ([137](#))

FpSigIsMember

Synopsis: Check whether a signal appears in a signal set.

Declaration: `function FpSigIsMember(const nset: TSigSet; signo: cInt) : cInt`

Visibility: default

Description: `FpSigIsMember` checks whether `SigNo` appears in the set `nset`. If it is a member, then 1 is returned. If not, zero is returned.

Errors: If an invalid signal number is given, -1 is returned.

See also: `FpSigEmptySet` (136), `FpSigFillSet` (136), `FpSigAddSet` (135), `FpSigDelSet` (136)

FpSignal

Synopsis: Install signal handler (deprecated)

Declaration: `function FpSignal(signum: LongInt; Handler: SignalHandler)
: SignalHandler`

Visibility: default

Description: `FpSignal` installs a new signal handler (specified by `Handler`) for signal `SigNum`.

This call has a subset of the functionality provided by the `FpSigAction` (134) call. The return value for `FpSignal` is the old signal handler, or nil on error.

Errors: Extended error information can be retrieved using `fpGetErrno` (112).

SIG_ERR An error occurred.

See also: `FpSigAction` (134), `FpKill` (116)

Listing: `./bunixex/ex58.pp`

Program `example58;`

{ Program to demonstrate the Signal function. }

*{
do a kill -USR1 pid from another terminal to see what happens.
replace pid with the real pid of this program.
You can get this pid by running 'ps'.
}*

uses `BaseUnix;`

Procedure `DoSig(sig : cint); cdecl;`

begin

`writeln('Receiving signal: ', sig);`
end;

begin

`if fpSignal(SigUsr1, SignalHandler(@DoSig)) = signalhandler(SIG_ERR) then`
begin
`writeln('Error: ', fpGetErrno, '.');`
`halt(1);`

```

    end;
    WriteIn ( 'Send USR1 signal or press <ENTER> to exit ');
    readIn;
end.

```

FpSigPending

Synopsis: Return set of currently pending signals

Declaration: `function FpSigPending(var nset: TSigSet) : cInt`

Visibility: default

Description: `fpSigPending` allows the examination of pending signals (which have been raised while blocked.)
The signal mask of pending signals is returned.

Errors: None

See also: `fpSigAction` ([134](#)), `fpSigProcMask` ([138](#)), `fpSigSuspend` ([138](#)), `fpSignal` ([137](#)), `fpKill` ([116](#))

FpSigProcMask

Synopsis: Set list of blocked signals

Declaration: `function FpSigProcMask(how: cInt;nset: PSigSet;oset: PSigSet) : cInt`
`function FpSigProcMask(how: cInt;const nset: TSigSet;var oset: TSigSet)`
`: cInt`

Visibility: default

Description: Changes the list of currently blocked signals. The behaviour of the call depends on How :

SIG_BLOCKThe set of blocked signals is the union of the current set and the `nset` argument.

SIG_UNBLOCKThe signals in `nset` are removed from the set of currently blocked signals.

SIG_SETMASKThe list of blocked signals is set so `nset`.

If `oset` is non-nil, then the old set is stored in it.

Errors: `Errno` is used to report errors.

sys_efault`oset` or `nset` point to an adress outside the range of the process.

sys_eintrSystem call was interrupted.

See also: `fpSigAction` ([134](#)), `fpSigPending` ([138](#)), `fpSigSuspend` ([138](#)), `fpKill` ([116](#))

FpSigSuspend

Synopsis: Set signal mask and suspend process till signal is received

Declaration: `function FpSigSuspend(const sigmask: TSigSet) : cInt`

Visibility: default

Description: `fpSigSuspend` temporarily replaces the signal mask for the process with the one given in `SigMask`,
and then suspends the process until a signal is received.

Errors: None

See also: `fpSigAction` ([134](#)), `fpSigProcMask` ([138](#)), `fpSigPending` ([138](#)), `fpSignal` ([137](#)), `fpKill` ([116](#))

FpSleep

Synopsis: Suspend process for several seconds

Declaration: `function FpSleep(seconds: cUInt) : cUInt`

Visibility: default

Description: `FpSleep` suspends the process till a time period as specified in `seconds` has passed, then the function returns. If the call was interrupted (e.g. by some signal) then the function may return earlier, and the return value is the remaining time till the end of the intended period.

If the function returns without error, the return value is zero.

See also: `fpPause` ([126](#)), `fpAlarm` ([100](#)), `fpNanoSleep` ([122](#))

Listing: `./bunixex/ex73.pp`

```

program example73;

  { Program to demonstrate the FpSleep function. }

uses BaseUnix;

Var
  Res : Longint;

begin
  Write('Sleep returned : ');
  Flush(Output);
  Res:=(fpSleep(10));
  Writeln(res);
  If (res<>0) then
    Writeln('Remaining seconds      : ',res);
end.

```

FpStat

Synopsis: Retrieve file information about a file descriptor.

Declaration: `function FpStat(path: pChar;var buf: Stat) : cInt`
`function FpStat(path: AnsiString;var buf: Stat) : cInt`
`function FpStat(path: String;var buf: Stat) : cInt`

Visibility: default

Description: `FpFStat` gets information about the file specified in `Path`, and stores it in `Info`, which is of type `stat` ([95](#)). The function returns zero if the call was succesfull, a nonzero return value indicates failure.

Errors: Extended error information can be retrieved using `fpGetErrno` ([112](#)).

`sys_enoentPath` does not exist.

See also: `FpStat` ([139](#)), `FpLStat` ([119](#))

Listing: `./bunixex/ex28.pp`

```

program example28;

{ Program to demonstrate the FStat function. }

uses BaseUnix;

var f : text;
    i : byte;
    info : stat;

begin
    { Make a file }
    assign (f, 'test.fil');
    rewrite (f);
    for i:=1 to 10 do writeln (f, 'Testline # ', i);
    close (f);
    { Do the call on made file. }
    if fpstat ('test.fil', info) <> 0 then
        begin
            writeln ('Fstat failed. Errno : ', fpgeterrno);
            halt (1);
        end;
    writeln;
    writeln ('Result of fstat on file ''test.fil''.');
    writeln ('Inode   : ', info.st_ino);
    writeln ('Mode    : ', info.st_mode);
    writeln ('nlink   : ', info.st_nlink);
    writeln ('uid     : ', info.st_uid);
    writeln ('gid     : ', info.st_gid);
    writeln ('rdev    : ', info.st_rdev);
    writeln ('Size    : ', info.st_size);
    writeln ('Blksize : ', info.st_blksize);
    writeln ('Blocks  : ', info.st_blocks);
    writeln ('atime   : ', info.st_atime);
    writeln ('mtime   : ', info.st_mtime);
    writeln ('ctime   : ', info.st_ctime);
    { Remove file }
    erase (f);
end.

```

fpSymlink

Synopsis: Create a symbolic link

Declaration: `function fpSymlink(oldname: pchar; newname: pchar) : cInt`

Visibility: default

Description: `SymLink` makes `NewName` point to the file in `OldName`, which doesn't necessarily exist. The two files DO NOT have the same inode number. This is known as a 'soft' link.

The permissions of the link are irrelevant, as they are not used when following the link. Ownership of the file is only checked in case of removal or renaming of the link.

The function returns zero if the call was succesful, a nonzero value if the call failed.

Errors: Extended error information is returned by the `FpGetErrno` ([112](#)) function.

sys_epermThe filesystem containing oldpath and newpath does not support linking files.

sys_eaccessWrite access for the directory containing Newpath is disallowed, or one of the directories in OldPath or NewPath has no search (=execute) permission.

sys_enoentA directory entry in OldPath or NewPath does not exist or is a symbolic link pointing to a non-existent directory.

sys_enotdirA directory entry in OldPath or NewPath is nor a directory.

sys_enomemInsufficient kernel memory.

sys_erofsThe files are on a read-only filesystem.

sys_eexistNewPath already exists.

sys_eloopOldPath or NewPath has a reference to a circular symbolic link, i.e. a symbolic link, whose expansion points to itself.

sys_enospcThe device containing NewPath has no room for another entry.

See also: FpLink (117), FpUnLink (145), FpReadLink (129)

Listing: ./unixex/ex22.pp

Program Example22;

{ Program to demonstrate the SymLink and UnLink functions. }

Uses baseunix, Unix;

Var F : Text;
S : String;

begin
Assign (F, 'test.txt');
Rewrite (F);
Writeln (F, 'This is written to test.txt');
Close(f);
{ new.txt and test.txt are now the same file }
if fpSymLink ('test.txt', 'new.txt') <> 0 **then**
 writeln ('Error when symlinking !');
{ Removing test.txt still leaves new.txt
Pointing now to a non-existent file ! }
If fpUnlink ('test.txt') <> 0 **then**
 Writeln ('Error when unlinking !');
Assign (f, 'new.txt');
{ This should fail, since the symbolic link
points to a non-existent file ! }
{\$i-}
Reset (F);
{\$i+}
If IOResult=0 **then**
 Writeln ('This shouldn''t happen');
{ Now remove new.txt also }
If fpUnlink ('new.txt') <> 0 **then**
 Writeln ('Error when unlinking !');
end.

fpS_ISBLK

Synopsis: Is file a block device

Declaration: `function fpS_ISBLK(m: TMode) : Boolean`

Visibility: default

Description: `fpS_ISBLK` checks the file mode `m` to see whether the file is a block device file. If so it returns `True`.

See also: `FpFStat` ([110](#)), `FpS_ISLNK` ([143](#)), `FpS_ISREG` ([143](#)), `FpS_ISDIR` ([142](#)), `FpS_ISCHR` ([142](#)), `FpS_ISFIFO` ([142](#)), `FpS_ISSOCK` ([144](#))

fpS_ISCHR

Synopsis: Is file a character device

Declaration: `function fpS_ISCHR(m: TMode) : Boolean`

Visibility: default

Description: `fpS_ISCHR` checks the file mode `m` to see whether the file is a character device file. If so it returns `True`.

See also: `FpFStat` ([110](#)), `FpS_ISLNK` ([143](#)), `FpS_ISREG` ([143](#)), `FpS_ISDIR` ([142](#)), `FpS_ISBLK` ([141](#)), `FpS_ISFIFO` ([142](#)), `FpS_ISSOCK` ([144](#))

fpS_ISDIR

Synopsis: Is file a directory

Declaration: `function fpS_ISDIR(m: TMode) : Boolean`

Visibility: default

Description: `fpS_ISDIR` checks the file mode `m` to see whether the file is a directory. If so, it returns `True`

See also: `FpFStat` ([110](#)), `FpS_ISLNK` ([143](#)), `FpS_ISREG` ([143](#)), `FpS_ISCHR` ([142](#)), `FpS_ISBLK` ([141](#)), `fpS_ISFIFO` ([142](#)), `FpS_ISSOCK` ([144](#))

fpS_ISFIFO

Synopsis: Is file a FIFO

Declaration: `function fpS_ISFIFO(m: TMode) : Boolean`

Visibility: default

Description: `fpS_ISFIFO` checks the file mode `m` to see whether the file is a fifo (a named pipe). If so it returns `True`.

See also: `FpFStat` ([110](#)), `FpS_ISLNK` ([143](#)), `FpS_ISREG` ([143](#)), `FpS_ISCHR` ([142](#)), `FpS_ISBLK` ([141](#)), `FpS_ISDIR` ([142](#)), `FpS_ISSOCK` ([144](#))

fpS_ISLNK

Synopsis: Is file a symbolic link

Declaration: `function fpS_ISLNK(m: TMode) : Boolean`

Visibility: default

Description: `fpS_ISLNK` checks the file mode `m` to see whether the file is a symbolic link. If so it returns `True`

See also: `FpFStat` (110), `FpS_ISFIFO` (142), `FpS_ISREG` (143), `FpS_ISCHR` (142), `FpS_ISBLK` (141), `FpS_ISDIR` (142), `FpS_ISSOCK` (144)

Listing: `./bunixex/ex53.pp`

Program `Example53;`

{ Program to demonstrate the S_ISLNK function. }

Uses `BaseUnix, Unix;`

Var `Info : Stat;`

begin

```

  if fpLStat (paramstr(1), @info)=0 then
    begin
      if fpS_ISLNK(info.st_mode) then
        Writeln ('File is a link');
      if fpS_ISREG(info.st_mode) then
        Writeln ('File is a regular file');
      if fpS_ISDIR(info.st_mode) then
        Writeln ('File is a directory');
      if fpS_ISCHR(info.st_mode) then
        Writeln ('File is a character device file');
      if fpS_ISBLK(info.st_mode) then
        Writeln ('File is a block device file');
      if fpS_ISFIFO(info.st_mode) then
        Writeln ('File is a named pipe (FIFO)');
      if fpS_ISSOCK(info.st_mode) then
        Writeln ('File is a socket');
    end;

```

end.

fpS_ISREG

Synopsis: Is file a regular file

Declaration: `function fpS_ISREG(m: TMode) : Boolean`

Visibility: default

Description: `fpS_ISREG` checks the file mode `m` to see whether the file is a regular file. If so it returns `True`

See also: `FpFStat` (110), `FpS_ISFIFO` (142), `FpS_ISLNK` (143), `FpS_ISCHR` (142), `FpS_ISBLK` (141), `FpS_ISDIR` (142), `FpS_ISSOCK` (144)

fpS_ISSOCK

Synopsis: Is file a unix socket

Declaration: `function fpS_ISSOCK(m: TMode) : Boolean`

Visibility: default

Description: `FpS_ISSOCK` checks the file mode `m` to see whether the file is a socket. If so it returns `True`.

See also: `FpFStat` (110), `FpS_ISFIFO` (142), `FpS_ISLNK` (143), `FpS_ISCHR` (142), `FpS_ISBLK` (141), `FpS_ISDIR` (142), `FpS_ISREG` (143)

fpTime

Synopsis: Return the current unix time

Declaration: `function FpTime(var tloc: TTime) : TTime`
`function fpTime : time_t`

Visibility: default

Description: `FpTime` returns the number of seconds since 00:00:00 GMT, january 1, 1970. it is adjusted to the local time zone, but not to DST. The result is also stored in `tloc`, if it is specified.

Errors: On error, -1 is returned. Extended error information can be retrieved using `fpGetErrno` (112).

Listing: `./bunixex/ex1.pp`

Program `Example1`;

{ Program to demonstrate the GetEpochTime function. }

Uses `Unix`;

begin

`Write` ('Secs past the start of the Epoch (00:00 1/1/1980) : ');

`WriteLn` (`GetEpochTime`);

end.

FpTimes

Synopsis: Return execution times for the current process

Declaration: `function FpTimes(var buffer: tms) : TClock`

Visibility: default

Description: `fpTimes` stores the execution time of the current process and child processes in `buffer`.

The return value (on linux) is the number of clock ticks since boot time. On error, -1 is returned, and extended error information can be retrieved with `fpGetErrno` (112).

See also: `fpUTime` (146)

FpUmask

Synopsis: Set file creation mask.

Declaration: `function FpUmask(cmask: TMode) : TMode`

Visibility: default

Description: `fpUmask` changes the file creation mask for the current user to `cmask`. The current mask is returned.

See also: `fpChmod` ([101](#))

Listing: `./bunixex/ex27.pp`

Program `Example27;`

{ Program to demonstrate the Umask function. }

Uses `BaseUnix;`

begin

WriteLn ('Old Umask was : ',fpUmask(&111));

WRitLn ('New Umask is : ',&111);

end.

FpUname

Synopsis: Return system name.

Declaration: `function FpUname(var name: UtsName) : cInt`

Visibility: default

Description: `Uname` gets the name and configuration of the current linux kernel, and returns it in the name record.

On success, 0 is returned, on error, -1 is returned.

Errors: Extended error information can be retrieved using `fpGetErrno` ([112](#)).

See also: `FpUTime` ([146](#))

FpUnlink

Synopsis: Unlink (i.e. remove) a file.

Declaration: `function FpUnlink(path: pChar) : cInt`
`function FpUnlink(path: AnsiString) : cInt`

Visibility: default

Description: `FpUnLink` decreases the link count on file `Path`. `Path` can be of type `AnsiString` or `PChar`. If the link count is zero, the file is removed from the disk.

The function returns zero if the call was succesfull, a nonzero value indicates failure.

For an example, see `FpLink` ([117](#)).

Errors: Extended error information can be retrieved using `fpGetErrno` ([112](#)).

sys_eaccessYou have no write access right in the directory containing Path, or you have no search permission in one of the directory components of Path.

sys_epermThe directory containing pathname has the sticky-bit set and the process's effective uid is neither the uid of the file to be deleted nor that of the directory containing it.

sys_enoentA component of the path doesn't exist.

sys_enotdirA directory component of the path is not a directory.

sys_eisdirPath refers to a directory.

sys_enomemInsufficient kernel memory.

sys_erofsPath is on a read-only filesystem.

See also: FpLink (117), FpSymLink (140)

FpUtime

Synopsis: Set access and modification times of a file (touch).

Declaration: `function FpUtime(path: pChar;times: pUtimBuf) : cInt`
`function FpUtime(path: AnsiString;times: pUtimBuf) : cInt`

Visibility: default

Description: FpUtime sets the access and modification times of the file specified in Path. the times record contains 2 fields, actime, and modtime, both of type time_t (commonly a longint). They should be filled with an epoch-like time, specifying, respectively, the last access time, and the last modification time. For some filesystem (most notably, FAT), these times are the same.

The function returns zero on success, a nonzero return value indicates failure.

Errors: Extended error information can be retrieved using fpGetErrno (112).

sys_eaccessOne of the directories in Path has no search (=execute) permission.

sys_enoentA directory entry in Path does not exist or is a symbolic link pointing to a non-existent directory.

Other errors may occur, but aren't documented.

See also: FpTime (144), FpChown (102), FpAccess (99)

Listing: ./bunixex/ex25.pp

Program Example25;

{ Program to demonstrate the UTime function . }

Uses BaseUnix , Unix , UnixUtil ;

Var utim : utimbuf ;
year , month , day , hour , minute , second : Word ;

begin
{ Set access and modification time of executable source }
GetTime (hour , minute , second) ;
GetDate (year , month , day) ;
utim . actime := LocalToEpoch (year , month , day , hour , minute , second) ;
utim . modtime := utim . actime ;
if Fputime ('ex25.pp' , @utim) <> 0 **then**

```

        writeln ( 'Call to UTime failed !' )
    else
        begin
            Write ( 'Set access and modification times to : ' );
            Write ( Hour:2, ':', minute:2, ':', second, ', ' );
            Writeln ( Day:2, '/', month:2, '/', year:4 );
        end;
    end.

```

FpWait

Synopsis: Wait for a child to exit.

Declaration: function FpWait(var stat_loc: cInt) : TPid

Visibility: default

Description: fpWait suspends the current process and waits for any child to exit or stop due to a signal. It reports the exit status of the exited child in stat_loc.

The return value of the function is the process ID of the child that exited, or -1 on error.

Errors: Extended error information can be retrieved using fpgetErrno ([112](#)).

See also: fpFork ([109](#)), fpExecve ([106](#)), fpWaitPid ([147](#))

FpWaitPid

Synopsis: Wait for a process to terminate

Declaration: function FpWaitpid(pid: TPid; stat_loc: pcInt; options: cInt) : TPid
 function FpWaitPid(pid: TPid; var Status: cInt; Options: cInt) : TPid

Visibility: default

Description: fpWaitPid waits for a child process with process ID Pid to exit. The value of Pid can be one of the following:

Pid < -1 Causes fpWaitPid to wait for any child process whose process group ID equals the absolute value of pid.

Pid = -1 Causes fpWaitPid to wait for any child process.

Pid = 0 Causes fpWaitPid to wait for any child process whose process group ID equals the one of the calling process.

Pid > 0 Causes fpWaitPid to wait for the child whose process ID equals the value of Pid.

The Options parameter can be used to specify further how fpWaitPid behaves:

WNOHANG Causes fpWaitpid to return immediately if no child has exited.

WUNTRACED Causes fpWaitPid to return also for children which are stopped, but whose status has not yet been reported.

__WCLONE Causes fpWaitPid also to wait for threads created by the #rtl.linux.Clone ([597](#)) call.

The exit status of the process that caused fpWaitPID is reported in stat_loc or Status.

Upon return, it returns the process id of the process that exited, 0 if no process exited, or -1 in case of failure.

For an example, see fpFork ([109](#)).

Errors: Extended error information can be retrieved using `fpgetErrno` ([112](#)).

See also: `fpFork` ([109](#)), `fpExecve` ([106](#)), `fpWait` ([147](#))

FpWrite

Synopsis: Write data to file descriptor

Declaration: `function FpWrite(fd: cInt;buf: pChar;nbytes: TSize) : TsSize`
`function FpWrite(fd: cInt;const buf;nbytes: TSize) : TsSize`

Visibility: default

Description: `FpWrite` writes at most `nbytes` bytes from `buf` to file descriptor `fd`.

The function returns the number of bytes actually written, or -1 if an error occurred.

Errors: Extended error information can be retrieved using `fpGetErrno` ([112](#)).

See also: `FpOpen` ([124](#)), `FpClose` ([103](#)), `FpRead` ([127](#)), `FpFTruncate` ([111](#)), `FpLSeek` ([118](#))

wexitStatus

Synopsis: Extract the exit status from the `fpWaitPID` ([147](#)) result.

Declaration: `function wexitStatus(Status: cInt) : cInt`

Visibility: default

Description: `WEXITSTATUS` can be used to extract the exit status from `Status`, the result of the `FpWaitPID` ([147](#)) call.

See also: `FpWaitPID` ([147](#)), `WTERMSIG` ([149](#)), `WSTOPSIG` ([149](#)), `WIFEXITED` ([148](#)), `WIFSIGNALED` ([148](#))

wifexited

Synopsis: Check whether the process exited normally

Declaration: `function wifexited(Status: cInt) : cInt`

Visibility: default

Description: `WIFEXITED` checks `Status` and returns `True` if the status indicates that the process terminated normally, i.e. was not stopped by a signal.

See also: `FpWaitPID` ([147](#)), `WTERMSIG` ([149](#)), `WSTOPSIG` ([149](#)), `WIFSIGNALED` ([148](#)), `WEXITSTATUS` ([148](#))

wifsignaled

Synopsis: Check whether the process was exited by a signal.

Declaration: `function wifsignaled(Status: cInt) : cInt`

Visibility: default

Description: `WIFSIGNALED` returns `True` if `Status` indicates that the process exited because it received a signal.

See also: [FpWaitPID \(147\)](#), [WTERMSIG \(149\)](#), [WSTOPSIG \(149\)](#), [WIFEXITED \(148\)](#), [WEXITSTATUS \(148\)](#)

wstopsig

Synopsis: Return the exit code from the process.

Declaration: `function wstopsig(Status: cInt) : cInt`

Visibility: default

Description: WSTOPSIG is an alias for WEXITSTATUS ([148](#)).

See also: [FpWaitPID \(147\)](#), [WTERMSIG \(149\)](#), [WIFEXITED \(148\)](#), [WIFSIGNALED \(148\)](#), [WEXITSTATUS \(148\)](#)

wtermsig

Synopsis: Return the signal that caused a process to exit.

Declaration: `function wtermsig(Status: cInt) : cInt`

Visibility: default

Description: WTERMSIG extracts from Status the signal number which caused the process to exit.

See also: [FpWaitPID \(147\)](#), [WSTOPSIG \(149\)](#), [WIFEXITED \(148\)](#), [WIFSIGNALED \(148\)](#), [WEXITSTATUS \(148\)](#)

Chapter 2

Reference for unit 'Classes'

2.1 Used units

Table 2.1: Used units by unit 'Classes'

Name	Page
rtlconst	150
sysutils	1071
types	150
typinfo	1203

2.2 Overview

This documentation describes the FPC `classes` unit. The `Classes` unit contains basic classes for the Free Component Library (FCL):

- a `TList` ([242](#)) class for maintaining lists of pointers,
- `TStringList` ([280](#)) for lists of strings,
- `TCollection` ([206](#)) to manage collections of objects
- `TStream` ([271](#)) classes to support streaming.

Furthermore it introduces methods for object persistence, and classes that understand an owner-owned relationship, with automatic memory management.

2.3 Constants, types and variables

Constants

`BITSHIFT` = 5

Used to calculate the size of a bits array

`FilerSignature : Array[1..4] of Char = 'TPF0'`

Constant that is found at the start of a binary stream containing a streamed component.

`fmCreate = $FFFF`

`TFileStream.Create (239)` creates a new file if needed.

`fmOpenRead = 0`

`TFileStream.Create (239)` opens a file with read-only access.

`fmOpenReadWrite = 2`

`TFileStream.Create (239)` opens a file with read-write access.

`fmOpenWrite = 1`

`TFileStream.Create (239)` opens a file with write-only access.

`MASK = 31`

Bitmask with all bits on.

`MaxBitFlags = MaxBitRec * 32`

Maximum number of bits in TBits collection.

`MaxBitRec = $FFFF div (SizeOf (longint))`

Maximum number of bit records in TBits.

`MaxListSize = Maxint div 16`

This constant sets the maximum number of elements in a TList (242).

`scAlt = $8000`

Indicates ALT key in a keyboard shortcut.

`scCtrl = $4000`

indicates CTRL key in a keyboard shortcut.

`scNone = 0`

Indicates no special key is presed in a keyboard shortcut.

`scShift = $2000`

Indicates Shift key in a keyboard shortcut.

`soFromBeginning = 0`

`Seek (273)` starts relative to the stream origin.

`soFromCurrent = 1`

Seek (273) starts relative to the current position in the stream.

`soFromEnd = 2`

Seek (273) starts relative to the stream end.

`toEOF = Char (0)`

Value returned by `TParser.Token` (255) when the end of the input stream was reached.

`toFloat = Char (4)`

Value returned by `TParser.Token` (255) when a floating point value was found in the input stream.

`toInteger = Char (3)`

Value returned by `TParser.Token` (255) when an integer was found in the input stream.

`toString = Char (2)`

Value returned by `TParser.Token` (255) when a string was found in the input stream.

`toSymbol = Char (1)`

Value returned by `TParser.Token` (255) when a symbol was found in the input stream.

Types

`HModule = System.HModule`

FPC doesn't support modules yet, so this is a dummy type.

`HRSRC = LongInt`

This type is provided for Delphi compatibility, it is used for resource streams.

`PPointerList = ^TPointerList`

Pointer to an array of pointers.

`PStringItem = ^TStringItem`

Pointer to a `TStringItem` (159) record.

`PStringItemList = ^TStringItemList`

Pointer to a `TStringItemList` (159).

`TActiveXRegType = (axrComponentOnly, axrIncludeDescendants)`

This type is provided for compatibility only, and is currently not used in Free Pascal.

Table 2.2: Enumeration values for type TActiveXRegType

Value	Explanation
axrComponentOnly	
axrIncludeDescendants	

Table 2.3: Enumeration values for type TAlignment

Value	Explanation
taCenter	Text is displayed centered.
taLeftJustify	Text is displayed aligned to the left
taRightJustify	Text is displayed aligned to the right.

```
TAlignment = (taLeftJustify, taRightJustify, taCenter)
```

The TAlignment type is used to specify the alignment of the text in controls that display a text.

```
TAncestorNotFoundEvent = procedure(Reader: TReader;
                                   const ComponentName: String;
                                   ComponentClass: TPersistentClass;
                                   var Component: TComponent) of object
```

This event occurs when an ancestor component cannot be found.

```
TBasicActionClass = Class of TBasicAction
```

TBasicAction ([184](#)) class reference.

```
TBasicActionLinkClass = Class of TBasicActionLink
```

TBasicActionLink ([189](#)) class reference.

```
TBitArray = Array[0..MaxBitRec-1] of cardinal
```

Array to store bits.

```
TCollectionItemClass = Class of TCollectionItem
```

TCollectionItemClass is used by the TCollection.ItemClass ([212](#)) property of TCollection ([206](#)) to identify the descendent class of TCollectionItem ([213](#)) which should be created and managed.

```
TComponentClass = Class of TComponent
```

The TComponentClass type is used when constructing TComponent ([217](#)) descendent instances and when registering components.

```
TComponentName = String
```

Names of components are of type TComponentName. By specifying a different type, the Object inspector can handle this property differently than a standard string property.

TComponentState= Set of (csLoading,csReading,csWriting,csDestroying,
csDesigning,csAncestor,csUpdating,csFixups,
csFreeNotification,csInline,csDesignInstance)

Indicates the state of the component during the streaming process.

TComponentStyle= Set of (csInheritable,csCheckPropAvail)

Describes the style of the component.

TCreateComponentEvent = procedure(Reader: TReader;
ComponentClass: TComponentClass;
var Component: TComponent) of object

Event handler type, occurs when a component instance must be created when a component is read from a stream.

TDuplicates = (dupIgnore,dupAccept,dupError)

Table 2.4: Enumeration values for type TDuplicates

Value	Explanation
dupAccept	Duplicate values can be added to the list.
dupError	If an attempt is made to add a duplicate value to the list, an EStringListError (174) exception is raised.
dupIgnore	Duplicate values will not be added to the list, but no error will be triggered.

Type to describe what to do with duplicate values in a TStringlist (280).

TFilerFlag = (ffInherited,ffChildPos,ffInline)

Table 2.5: Enumeration values for type TFilerFlag

Value	Explanation
ffChildPos	The position of the child on it's parent is included.
ffInherited	Stored object is an inherited object.
ffInline	Used for frames.

The TFiler class uses this enumeration type to decide whether the streamed object was streamed as part of an inherited form or not.

TFilerFlags= Set of (ffChildPos,ffInherited,ffInline)

Set of TFilerFlag (154)

TFindAncestorEvent = procedure(Writer: TWriter;Component: TComponent;
const Name: String;
var Ancestor: TComponent;
var RootAncestor: TComponent) of object

Event that occurs w

```
TFindComponentClassEvent = procedure(Reader: TReader;
                                     const ClassName: String;
                                     var ComponentClass: TComponentClass)
                                     of object
```

Event handler type, occurs when a component class pointer must be found when reading a component from a stream.

```
TFindGlobalComponent = function(const Name: String) : TComponent
```

TFindGlobalComponent is a callback used to find a component in a global scope. It is used when the streaming system needs to find a component which is not part of the component which is currently being streamed. It should return the component with name Name, or Nil if none is found.

The variable FindGlobalComponent ([161](#)) is a callback of type TFindGlobalComponent. It can be set by the IDE when an unknown reference is found, to offer the designer to redirect the link to a new component.

```
TFindMethodEvent = procedure(Reader: TReader;const MethodName: String;
                             var Address: Pointer;var Error: Boolean)
                             of object
```

If a TReader ([258](#)) instance needs to locate a method and it doesn't find it in the streamed form, then the OnFindMethod ([267](#)) event handler will be called, if one is installed. This event can be assigned in order to use different locating methods. If a method is found, then its address should be returned in Address. The Error should be set to True if the reader should raise an exception after the event was handled. If it is set to False no exception will be raised, even if no method was found. On entry, Error will be set to True.

```
TGetChildProc = procedure(Child: TComponent) of object
```

Callback used when obtaining child components.

```
TGetStrProc = procedure(const S: String) of object
```

This event is used as a callback to retrieve string values. It is used, among other things, to pass along string properties in property editors.

```
THandle = System.THandle
```

This type is used as the handle for THandleStream ([240](#)) stream descendents

```
THelpContext = -MaxLongint..MaxLongint
```

Range type to specify help contexts.

```
THelpEvent = function(Command: Word;Data: LongInt;var CallHelp: Boolean)
                 : Boolean of object
```

This event is used for display of online help.

```
THelpType = (htKeyword,htContext)
```

Enumeration type specifying the kind of help requested.

Table 2.6: Enumeration values for type THelpType

Value	Explanation
htContext	
htKeyword	

```
TIdentMapEntry = record
  Value : Integer;
  Name : String;
end
```

TIdentMapEntry is used internally by the `IdentToInt` (164) and `IntToIdent` (164) calls to store the mapping between the identifiers and the integers they represent.

```
TIdentToInt = function(const Ident: String;var Int: LongInt) : Boolean
```

TIdentToInt is a callback used to look up identifiers (Ident) and return an integer value corresponding to this identifier (Int). The callback should return True if a value corresponding to integer Ident was found, False if not.

A callback of type TIdentToInt should be specified when an integer is registered using the `RegisterIntegerConsts` (169) call.

```
TInitComponentHandler = function(Instance: TComponent;
                                   RootAncestor: TClass) : Boolean
```

```
TIntToIdent = function(Int: LongInt;var Ident: String) : Boolean
```

TIntToIdent is a callback used to look up integers (Ident) and return an identifier (Ident) that can be used to represent this integer value in an IDE. The callback should return True if a value corresponding to integer Ident was found, False if not.

A callback of type TIntToIdent should be specified when an integer is registered using the `RegisterIntegerConsts` (169) call.

```
TListNotification = (lnAdded,lnExtracted,lnDeleted)
```

Table 2.7: Enumeration values for type TListNotification

Value	Explanation
lnAdded	
lnDeleted	
lnExtracted	

Kind of list notification event.

```
TListSortCompare = function(Item1: Pointer;Item2: Pointer) : Integer
```

Callback type for the list sort algorithm.

TNotifyEvent = procedure(Sender: TObject) of object

Most event handlers are implemented as a property of type TNotifyEvent. When this is set to a certain method of a class, when the event occurs, the method will be called, and the class that generated the event will pass itself along as the Sender argument.

TOperation = (opInsert,opRemove)

Table 2.8: Enumeration values for type TOperation

Value	Explanation
opInsert	A new component is being inserted in the child component list.
opRemove	A component is being removed from the child component list.

Operation of which a component is notified.

TPersistentClass = Class of TPersistent

TPersistentClass is the class reference type for the TPersistent (255) class.

TPoint = Types.TPoint

This record describes a coordinate. It is used to handle the Top (217) and Left (217) properties of TComponent (217).

X represents the X-Coordinate of the point described by the record. Y represents the Y-Coordinate of the point described by the record.

TPointerList = Array[0..MaxListSize-1] of Pointer

Type for an Array of pointers.

```
TPropertyNotFoundEvent = procedure(Reader: TReader;
                                   Instance: TPersistent;
                                   var PropName: String;IsPath: Boolean;
                                   var Handled: Boolean;
                                   var Skip: Boolean) of object
```

TReadComponentsProc = procedure(Component: TComponent) of object

Callback type when reading a component from a stream

```
TReaderError = procedure(Reader: TReader;const Message: String;
                        var Handled: Boolean) of object
```

Event handler type, called when an error occurs during the streaming.

TReaderProc = procedure(Reader: TReader) of object

The TReaderProc reader procedure is a callback procedure which will be used by a TPersistent (255) descendent to read user properties from a stream during the streaming process. The Reader argument is the writer object which can be used read properties from the stream.

```
TReadWriteStringPropertyEvent = procedure(Sender: TObject;
                                         const Instance: TPersistent;
                                         PropInfo: PPropInfo;
                                         var Content: String) of object
```

```
TRect = Types.TRect
```

TRect describes a rectangle in space with its upper-left (in (Top,Left>)) and lower-right (in (Bottom,Right)) corners.

```
TReferenceNameEvent = procedure(Reader: TReader;var Name: String)
                             of object
```

Occurs when a named object needs to be looked up.

```
TSeekOrigin = (soBeginning,soCurrent,soEnd)
```

Table 2.9: Enumeration values for type TSeekOrigin

Value	Explanation
soBeginning	Offset is interpreted relative to the start of the stream.
soCurrent	Offset is interpreted relative to the current position in the stream.
soEnd	Offset is interpreted relative to the end of the stream.

Specifies the origin of the TStream.Seek (273) method.

```
TSetMethodPropertyEvent = procedure(Reader: TReader;
                                     Instance: TPersistent;
                                     PropInfo: PPropInfo;
                                     const TheMethodName: String;
                                     var Handled: Boolean) of object
```

```
TSetNameEvent = procedure(Reader: TReader;Component: TComponent;
                          var Name: String) of object
```

Occurs when the reader needs to set a component's name.

```
TShiftState= Set of (ssShift,ssAlt,ssCtrl,ssLeft,ssRight,ssMiddle,
                     ssDouble,ssMeta,ssSuper,ssHyper,ssAltGr,ssCaps,
                     ssNum,ssScroll,ssTriple,ssQuad)
```

This type is used when describing a shortcut key or when describing what special keys are pressed on a keyboard when a key event is generated.

The set contains the special keys that can be used in combination with a 'normal' key.

```
TShortCut = ( Word )..High ( Word )
```

Enumeration type to identify shortcut key combinations.

```
TSmallPoint = Windows.TSmallPoint
```

Same as TPoint (157), only the X and Y ranges are limited to 2-byte integers instead of 4-byte integers.

TStreamProc = procedure(Stream: TStream) of object

Procedure type used in streaming.

```
TStringItem = record
  FString : String;
  FObject : TObject;
end
```

The TStringItem is used to store the string and object items in a TStringList (280) string list instance. It should never be used directly.

TStringItemList = Array[0..MaxListSize] of TStringItem

This declaration is provided for Delphi compatibility, it is not used in Free Pascal.

```
TStringListSortCompare = function(List: TStringList; Index1: Integer;
                                   Index2: Integer) : Integer
```

Callback type used in stringlist compares.

TSynchronizeProcVar = procedure

TThreadMethod = procedure of object

Procedure variable used when synchronizing threads.

```
TThreadPriority = (tpIdle, tpLowest, tpLower, tpNormal, tpHigher, tpHighest,
                   tpTimeCritical)
```

Table 2.10: Enumeration values for type TThreadPriority

Value	Explanation
tpHigher	Thread runs at high priority
tpHighest	Thread runs at highest possible priority.
tpIdle	Thread only runs when other processes are idle.
tpLower	Thread runs at a lower priority.
tpLowest	Thread runs at the lowest priority.
tpNormal	Thread runs at normal process priority.
tpTimeCritical	Thread runs at realtime priority.

Enumeration specifying the priority at which a thread runs.

```
TValueType = (vaNull, vaList, vaInt8, vaInt16, vaInt32, vaExtended, vaString,
              vaIdent, vaFalse, vaTrue, vaBinary, vaSet, vaLString, vaNil,
              vaCollection, vaSingle, vaCurrency, vaDate, vaWString, vaInt64)
```

Enumerated type used to identify the kind of streamed property

Table 2.11: Enumeration values for type TValueType

Value	Explanation
vaBinary	Binary data follows.
vaCollection	Collection follows
vaCurrency	Currency value follows
vaDate	Date value follows
vaExtended	Extended value.
vaFalse	Boolean False value.
vaIdent	Identifier.
vaInt16	Integer value, 16 bits long.
vaInt32	Integer value, 32 bits long.
vaInt64	Integer value, 64 bits long.
vaInt8	Integer value, 8 bits long.
vaList	Identifies the start of a list of values
vaLString	Ansistring data follows.
vaNil	Nil pointer.
vaNull	Empty value. Ends a list.
vaSet	Set data follows.
vaSingle	Single type follows.
vaString	String value.
vaTrue	Boolean True value.
vaWString	Widestring value follows.

```
TWriteMethodPropertyEvent = procedure(Writer: TWriter;
                                     Instance: TPersistent;
                                     PropInfo: PPropInfo;
                                     const MethodValue: TMethod;
                                     const DefMethodCodeValue: Pointer;
                                     var Handled: Boolean) of object
```

```
TWriterProc = procedure(Writer: TWriter) of object
```

The TWriterProc writer procedure is a callback procedure which will be used by a TPersistent (255) descendent to write user properties from a stream during the streaming process. The Writer argument is the writer object which can be used write properties to the stream.

Variables

```
AddDataModule : procedure(DataModule: TDataModule) of object
```

```
ApplicationHandleException : procedure(Sender: TObject) of object
```

```
ApplicationShowException : procedure(E: Exception) of object
```

```
FindGlobalComponent : TFindGlobalComponent
```

`FindGlobalComponent` is a callback of type `TFindGlobalComponent` (155). It can be set by the IDE when an unknown reference is found, to offer the user to redirect the link to a new component.

It is a callback used to find a component in a global scope. It is used when the streaming system needs to find a component which is not part of the component which is currently being streamed. It should return the component with name `Name`, or `Nil` if none is found.

`MainThreadID` : `THandle`

ID of main thread. Unused at this point.

`RegisterComponentsProc` : `procedure(const Page: String;
ComponentClasses: Array[] of TComponentClass)`

`RegisterComponentsProc` can be set by an IDE to be notified when new components are being registered. Application programmers should never have to set `RegisterComponentsProc`

`RegisterNoIconProc` : `procedure(ComponentClasses: Array[] of TComponentClass)`

`RegisterNoIconProc` can be set by an IDE to be notified when new components are being registered, and which do not need an Icon in the component palette. Application programmers should never have to set `RegisterComponentsProc`

`RemoveDataModule` : `procedure(DataModule: TDataModule) of object`

`SynchronizeMethodProc` : `TSynchronizeProcVar = nil`

2.4 Procedures and functions

BeginGlobalLoading

Synopsis: Not yet implemented

Declaration: `procedure BeginGlobalLoading`

Visibility: default

Description: Not yet implemented

Bounds

Synopsis: Returns a `TRect` structure with the bounding rect of the given location and size.

Declaration: `function Bounds(ALeft: Integer; ATop: Integer; AWidth: Integer;
AHeight: Integer) : TRect`

Visibility: default

Description: `Bounds` returns a `TRect` (158) record with the given origin (`ALeft`, `ATop`) and dimensions (`AWidth`, `AHeight`) filled in.

CollectionsEqual

Synopsis: Returns True if two collections are equal.

Declaration: `function CollectionsEqual(C1: TCollection; C2: TCollection) : Boolean`

Visibility: default

Description: `CollectionsEqual` is not yet implemented. It simply returns `False`

EndGlobalLoading

Synopsis: Not yet implemented.

Declaration: `procedure EndGlobalLoading`

Visibility: default

Description: Not yet implemented.

FindClass

Synopsis: Returns the class pointer of a class with given name.

Declaration: `function FindClass(const AClassName: String) : TPersistentClass`

Visibility: default

Description: `FindClass` searches for the class named `ClassName` in the list of registered classes and returns a class pointer to the definition. If no class with the given name could be found, an exception is raised.

The `GetClass` ([162](#)) function does not raise an exception when it does not find the class, but returns a `Nil` pointer instead.

See also: `RegisterClass` ([167](#)), `GetClass` ([162](#))

FindNestedComponent

Synopsis: Finds the component with name path starting at the indicated root component.

Declaration: `function FindNestedComponent(Root: TComponent; const NamePath: String)
: TComponent`

Visibility: default

Description: `FindNestedComponent` will descend through the list of owned components (starting at `Root`) and will return the component whose name path matches `NamePath`. As a path separator the characters `.` (dot), `-` (dash) and `>` (greater than) can be used

See also: `GlobalFixupReferences` ([163](#))

GetClass

Synopsis: Returns the class pointer of a class with given name.

Declaration: `function GetClass(const AClassName: String) : TPersistentClass`

Visibility: default

Description: `GetClass` searches for the class named `ClassName` in the list of registered classes and returns a class pointer to the definition. If no class with the given name could be found, `Nil` is returned.

The `FindClass` ([162](#)) function will raise an exception if the does not find the class.

See also: `RegisterClass` ([167](#)), `GetClass` ([162](#))

GetFixupInstanceNames

Synopsis: Returns the names of elements that need to be resolved for the `root` component, whose reference contains `ReferenceRootName`

Declaration:

```
procedure GetFixupInstanceNames(Root: TComponent;
                                const ReferenceRootName: String;
                                Names: TStrings)
```

Visibility: `default`

Description: `GetFixupInstanceNames` examines the list of unresolved references and returns the names of classes that contain unresolved references to the `Root` component in the list `Names`. The list is not cleared prior to filling it.

See also: `GetFixupReferenceNames` ([163](#)), `GlobalFixupReferences` ([163](#))

GetFixupReferenceNames

Synopsis: Returns the names of elements that need to be resolved for the `root` component.

Declaration:

```
procedure GetFixupReferenceNames(Root: TComponent; Names: TStrings)
```

Visibility: `default`

Description: `GetFixupReferenceNames` examines the list of unresolved references and returns the names of properties that must be resolved for the component `Root` in the list `Names`. The list is not cleared prior to filling it.

See also: `GetFixupInstanceNames` ([163](#)), `GlobalFixupReferences` ([163](#))

GlobalFixupReferences

Synopsis: Called to resolve unresolved references after forms are loaded.

Declaration:

```
procedure GlobalFixupReferences
```

Visibility: `default`

Description: `GlobalFixupReferences` runs over the list of unresolved references and tries to resolve them. This routine should under normal circumstances not be called in an application programmer's code. It is called automatically by the streaming system after a component has been instantiated and its properties read from a stream. It will attempt to resolve references to other global components.

See also: `GetFixupReferenceNames` ([163](#)), `GetFixupInstanceNames` ([163](#))

IdentToInt

Synopsis: Looks up an integer value in a integer-to-identifier map list.

Declaration: `function IdentToInt(const Ident: String; var Int: LongInt;
const Map: Array[] of TIdentMapEntry) : Boolean`

Visibility: default

Description: `IdentToInt` searches `Map` for an entry whose `Name` field matches `Ident` and returns the corresponding integer value in `Int`. If a match was found, the function returns `True`, otherwise, `False` is returned.

See also: `TIdentToInt` (156), `TIntToIdent` (156), `IntToIdent` (164), `TIdentMapEntry` (156)

InitComponentRes

Synopsis: Provided for Delphi compatibility only

Declaration: `function InitComponentRes(const ResName: String; Instance: TComponent)
: Boolean`

Visibility: default

Description: This function is provided for Delphi compatibility. It always returns `false`.

See also: `ReadComponentRes` (166)

InitInheritedComponent

Synopsis: Initializes a component descending from `RootAncestor`

Declaration: `function InitInheritedComponent(Instance: TComponent;
RootAncestor: TClass) : Boolean`

Visibility: default

Description: `InitInheritedComponent` should be called from a constructor to read properties of the component `Instance` from the streaming system. The `RootAncestor` class is the root class from which `Instance` is a descendent. This must be one of `TDatamodule`, `TCustomForm` or `TFrame`. The function returns `True` if the properties were successfully read from a stream or `False` if some error occurred.

See also: `ReadComponentRes` (166), `ReadComponentResEx` (166), `ReadComponentResFile` (166)

IntToIdent

Synopsis: Looks up an identifier for an integer value in a identifier-to-integer map list.

Declaration: `function IntToIdent(Int: LongInt; var Ident: String;
const Map: Array[] of TIdentMapEntry) : Boolean`

Visibility: default

Description: `IdentToInt` searches `Map` for an entry whose `Value` field matches `Int` and returns the corresponding identifier in `Ident`. If a match was found, the function returns `True`, otherwise, `False` is returned.

See also: `TIdentToInt` (156), `TIntToIdent` (156), `IdentToInt` (164), `TIdentMapEntry` (156)

LineStart

Synopsis: Finds the start of a line in `Buffer` before `BufPos`.

Declaration: `function LineStart(Buffer: PChar; BufPos: PChar) : PChar`

Visibility: default

Description: `LineStart` reversely scans `Buffer` starting at `BufPos` for a linefeed character. It returns a pointer at the linefeed character.

NotifyGlobalLoading

Synopsis: Not yet implemented.

Declaration: `procedure NotifyGlobalLoading`

Visibility: default

Description: Not yet implemented.

ObjectBinaryToText

Synopsis: Converts an object stream from a binary to a text format.

Declaration: `procedure ObjectBinaryToText(Input: TStream; Output: TStream)`

Visibility: default

Description: `ObjectBinaryToText` reads an object stream in binary format from `Input` and writes the object stream in text format to `Output`. No components are instantiated during the process, this is a pure conversion routine.

See also: `ObjectTextToBinary` ([165](#))

ObjectResourceToText

Synopsis: Converts an object stream from a (windows) resource to a text format.

Declaration: `procedure ObjectResourceToText(Input: TStream; Output: TStream)`

Visibility: default

Description: `ObjectResourceToText` reads the resource header from the `Input` stream and then passes the streams to `ObjectBinaryToText` ([165](#))

See also: `ObjectBinaryToText` ([165](#)), `ObjectTextToResource` ([166](#))

ObjectTextToBinary

Synopsis: Converts an object stream from a text to a binary format.

Declaration: `procedure ObjectTextToBinary(Input: TStream; Output: TStream)`

Visibility: default

Description: Converts an object stream from a text to a binary format.

ObjectTextToResource

Synopsis: Converts an object stream from a text to a (windows) resource format.

Declaration: `procedure ObjectTextToResource(Input: TStream;Output: TStream)`

Visibility: default

Description: `ObjectTextToResource` reads an object stream in text format from `Input` and writes a resource stream to `Output`.

Note that for the current implementation of this method in Free Pascal, the output stream should support positioning. (e.g. it should not be a pipe)

See also: `ObjectBinaryToText` ([165](#)), `ObjectResourceToText` ([165](#))

Point

Synopsis: Returns a `TPoint` record with the given coordinates.

Declaration: `function Point(AX: Integer;AY: Integer) : TPoint`

Visibility: default

Description: `Point` returns a `TPoint` ([157](#)) record with the given coordinates `AX` and `AY` filled in.

See also: `TPoint` ([157](#)), `SmallPoint` ([170](#)), `Rect` ([167](#)), `Bounds` ([161](#))

ReadComponentRes

Synopsis: Read component properties from a resource in the current module

Declaration: `function ReadComponentRes(const ResName: String;Instance: TComponent)
: TComponent`

Visibility: default

Description: This function is provided for Delphi compatibility. It always returns `Nil`.

ReadComponentResEx

Synopsis: Read component properties from a resource in the specified module

Declaration: `function ReadComponentResEx(HInstance: THandle;const ResName: String)
: TComponent`

Visibility: default

Description: This function is provided for Delphi compatibility. It always returns `Nil`.

ReadComponentResFile

Synopsis: Read component properties from a specified resource file

Declaration: `function ReadComponentResFile(const FileName: String;
Instance: TComponent) : TComponent`

Visibility: default

Description: `ReadComponentResFile` starts reading properties for `Instance` from the file `FileName`. It creates a filestream from `FileName` and then calls the `TStream.ReadComponentRes` (275) method to read the state of the component from the stream.

See also: `TStream.ReadComponentRes` (275), `WriteComponentResFile` (171)

Rect

Synopsis: Returns a `TRect` record with the given coordinates.

Declaration: `function Rect(ALeft: Integer; ATop: Integer; ARight: Integer; ABottom: Integer) : TRect`

Visibility: default

Description: `Rect` returns a `TRect` (158) record with the given top-left (`ALeft`, `ATop`) and bottom-right (`ABottom`, `ARight`) corners filled in.

No checking is done to see whether the coordinates are valid.

See also: `TRect` (158), `Point` (166), `SmallPoint` (170), `Bounds` (161)

RedirectFixupReferences

Synopsis: Redirects references under the root object from `OldRootName` to `NewRootName`

Declaration: `procedure RedirectFixupReferences(Root: TComponent; const OldRootName: String; const NewRootName: String)`

Visibility: default

Description: `RedirectFixupReferences` examines the list of unresolved references and replaces references to a root object named `OldRootName` with references to root object `NewRootName`.

An application programmer should never need to call `RedirectFixupReferences`. This function can be used by an IDE to support redirection of broken component links.

See also: `RemoveFixupReferences` (169)

RegisterClass

Synopsis: Registers a class with the streaming system.

Declaration: `procedure RegisterClass(AClass: TPersistentClass)`

Visibility: default

Description: `RegisterClass` registers the class `AClass` in the streaming system. After the class has been registered, it can be read from a stream when a reference to this class is encountered.

See also: `RegisterClasses` (168), `RegisterClassAlias` (168), `RegisterComponents` (168), `UnregisterClass` (170)

RegisterClassAlias

Synopsis: Registers a class alias with the streaming system.

Declaration: `procedure RegisterClassAlias(AClass: TPersistentClass;
const Alias: String)`

Visibility: default

Description: `RegisterClassAlias` registers a class alias in the streaming system. If a reference to a class `Alias` is encountered in a stream, then an instance of the class `AClass` will be created instead by the streaming code.

See also: `RegisterClass` ([167](#)), `RegisterClasses` ([168](#)), `RegisterComponents` ([168](#)), `UnregisterClass` ([170](#))

RegisterClasses

Synopsis: Registers multiple classes with the streaming system.

Declaration: `procedure RegisterClasses(AClasses: Array[] of TPersistentClass)`

Visibility: default

Description: `RegisterClasses` registers the specified classes `AClass` in the streaming system. After the classes have been registered, they can be read from a stream when a reference to this class is encountered.

See also: `RegisterClass` ([167](#)), `RegisterClassAlias` ([168](#)), `RegisterComponents` ([168](#)), `UnregisterClass` ([170](#))

RegisterComponents

Synopsis: Registers components for the component palette.

Declaration: `procedure RegisterComponents(const Page: String;
ComponentClasses: Array[] of TComponentClass)`

Visibility: default

Description: `RegisterComponents` registers the component on the appropriate component page. The component pages can be used by an IDE to display the known components so an application programmer may pick and use the components in his programs.

`Registercomponents` inserts the component class in the correct component page. If the `RegisterComponentsProc` procedure is set, this is called as well. Note that this behaviour is different from Delphi's behaviour where an exception will be raised if the procedural variable is not set.

See also: `RegisterClass` ([167](#)), `RegisterNoIcon` ([169](#))

RegisterInitComponentHandler

Declaration: `procedure RegisterInitComponentHandler(ComponentClass: TComponentClass;
Handler: TInitComponentHandler)`

Visibility: default

RegisterIntegerConsts

Synopsis: Registers some integer-to-identifier mappings.

Declaration: `procedure RegisterIntegerConsts(IntegerType: Pointer;
IdentToIntFn: TIdentToInt;
IntToIdentFn: TIntToIdent)`

Visibility: default

Description: `RegisterIntegerConsts` registers a pair of callbacks to be used when an integer of type `IntegerType` must be mapped to an identifier (using `IntToIdentFn`) or when an identifier must be mapped to an integer (using `IdentToIntFn`).

Component programmers can use `RegisterIntegerConsts` to associate a series of identifier strings with integer values for a property. A necessary condition is that the property should have a separate type declared using the `type integer` syntax. If a type of integer is defined in this way, an IDE can show symbolic names for the values of these properties.

The `IntegerType` should be a pointer to the type information of the integer type. The `IntToIdentFn` and `IdentToIntFn` are two callbacks that will be used when converting between the identifier and integer value and vice versa. The functions `IdentToInt` (164) and `IntToIdent` (164) can be used to implement these callback functions.

See also: `TIdentToInt` (156), `TIntToIdent` (156), `IdentToInt` (164), `IntToIdent` (164)

RegisterNoIcon

Synopsis: Registers components that have no icon on the component palette.

Declaration: `procedure RegisterNoIcon(ComponentClasses: Array[] of TComponentClass)`

Visibility: default

Description: `RegisterNoIcon` performs the same function as `RegisterComponents` (168) except that it calls `RegisterNoIconProc` (161) instead of `RegisterComponentsProc` (161)

See also: `RegisterNoIconProc` (161), `RegisterComponents` (168)

RegisterNonActiveX

Synopsis: Register non-activex component.

Declaration: `procedure RegisterNonActiveX
(ComponentClasses: Array[] of TComponentClass;
AxRegType: TActiveXRegType)`

Visibility: default

Description: Not yet implemented in Free Pascal

RemoveFixupReferences

Synopsis: Removes references to rootname from the fixup list.

Declaration: `procedure RemoveFixupReferences(Root: TComponent; const RootName: String)`

Visibility: default

Description: `RemoveFixupReferences` examines the list of unresolved references and removes references to a root object pointing at `Root` or a root component named `RootName`.

An application programmer should never need to call `RemoveFixupReferences`. This function can be used by an IDE to support removal of broken component links.

See also: `RedirectFixupReferences` ([167](#))

RemoveFixups

Synopsis: Removes `Instance` from the fixup list.

Declaration: `procedure RemoveFixups(Instance: TPersistent)`

Visibility: `default`

Description: `RemoveFixups` removes all entries for component `Instance` from the list of unresolved references.

See also: `RedirectFixupReferences` ([167](#)), `RemoveFixupReferences` ([169](#))

SmallPoint

Synopsis: Returns a `TSmallPoint` record with the given coordinates.

Declaration: `function SmallPoint(AX: SmallInt; AY: SmallInt) : TSmallPoint`

Visibility: `default`

Description: `SmallPoint` returns a `TSmallPoint` ([159](#)) record with the given coordinates `AX` and `AY` filled in.

See also: `TSmallPoint` ([159](#)), `Point` ([166](#)), `Rect` ([167](#)), `Bounds` ([161](#))

UnRegisterClass

Synopsis: Unregisters a class from the streaming system.

Declaration: `procedure UnRegisterClass(AClass: TPersistentClass)`

Visibility: `default`

Description: `UnregisterClass` removes the class `AClass` from the class definitions in the streaming system.

See also: `UnRegisterClasses` ([170](#)), `UnRegisterModuleClasses` ([171](#)), `RegisterClass` ([167](#))

UnRegisterClasses

Synopsis: Unregisters multiple classes from the streaming system.

Declaration: `procedure UnRegisterClasses(AClasses: Array[] of TPersistentClass)`

Visibility: `default`

Description: `UnregisterClasses` removes the classes in `AClasses` from the class definitions in the streaming system.

UnRegisterModuleClasses

Synopsis: Unregisters classes registered by module.

Declaration: `procedure UnRegisterModuleClasses(Module: HModule)`

Visibility: default

Description: `UnRegisterModuleClasses` unregisters all classes which reside in the module `Module`. For each registered class, the definition pointer is checked to see whether it resides in the module, and if it does, the definition is removed.

See also: `UnRegisterClass` ([170](#)), `UnRegisterClasses` ([170](#)), `RegisterClasses` ([168](#))

WriteComponentResFile

Synopsis: Write component properties to a specified resource file

Declaration: `procedure WriteComponentResFile(const FileName: String;
Instance: TComponent)`

Visibility: default

Description: `WriteComponentResFile` starts writing properties of `Instance` to the file `FileName`. It creates a filestream from `FileName` and then calls `TStream.WriteComponentRes` ([276](#)) method to write the state of the component to the stream.

See also: `TStream.WriteComponentRes` ([276](#)), `ReadComponentResFile` ([166](#))

2.5 EBitsError

Description

When an index of a bit in a `TBits` ([199](#)) is out of the valid range (0 to `Count-1`) then a `EBitsError` exception is raised.

2.6 EClassNotFound

Description

When the streaming system needs to create a component, it looks for the class pointer (VMT) in the list of registered classes by its name. If this name is not found, then an `EClassNotFound` is raised.

2.7 EComponentError

Description

When an error occurs during the registration of a component, or when naming a component, then a `EComponentError` is raised. Possible causes are:

1. An name with an illegal character was assigned to a component.
2. A component with the same name and owner already exists.
3. The component registration system isn't set up properly.

2.8 EFCREATEERROR

Description

When the operating system reports an error during creation of a new file in the Filestream Constructor (239), a `EFCREATEERROR` is raised.

2.9 EFILERERROR

Description

This class serves as an ancestor class for exceptions that are raised when an error occurs during component streaming. A `EfilerError` exception is raised when a class is registered twice.

2.10 EFOPENERROR

Description

When the operating system reports an error during the opening of a file in the Filestream Constructor (239), a `EFOPENERROR` is raised.

2.11 EINVALIDIMAGE

Description

This exception is not used by Free Pascal but is provided for Delphi compatibility.

2.12 EINVALIDOPERATION

Description

This exception is not used in Free Pascal, it is defined for Delphi compatibility purposes only.

2.13 ELISTERROR

Description

If an error occurs in one of the `TList` (242) or `TStrings` (287) methods, then a `ELISTERROR` exception is raised. This can occur in one of the following cases:

1. There is not enough memory to expand the list.
2. The list tried to grow beyond its maximal capacity.
3. An attempt was made to reduce the capacity of the list below the current element count.
4. An attempt was made to set the list count to a negative value.
5. A non-existent element of the list was referenced. (i.e. the list index was out of bounds)
6. An attempt was made to move an item to a position outside the list's bounds.

2.14 EMethodNotFound

Description

This exception is no longer used in the streaming system. This error is replaced by a EReadError ([173](#)).

2.15 EOutOfResources

Description

This exception is not used in Free Pascal, it is defined for Delphi compatibility purposes only.

2.16 EParserError

Description

When an error occurs during the parsing of a stream, an EParserError is raised. Usually this indicates that an invalid token was found on the input stream, or the token read from the stream wasn't the expected token.

2.17 EReadError

Description

If an error occurs when reading from a stream, a EReadError exception is raised. Possible causes for this are:

1. Not enough data is available when reading from a stream
2. The stream containing a component's data contains invalid data. this will occur only when reading a component from a stream.

2.18 EResNotFound

Description

This exception is not used by Free Pascal but is provided for Delphi compatibility.

2.19 EStreamError

Description

An EStreamError is raised when an error occurs during reading from or writing to a stream: Possible causes are

1. Not enough data is available in the stream.
2. Trying to seek beyond the beginning or end of the stream.

3. Trying to set the capacity of a memory stream and no memory is available.
4. Trying to write to a resource stream.

2.20 TStringListError

Description

When an error occurs in one of the methods of `TStrings` (287) then an `EStringListError` is raised. This can have one of the following causes:

1. There is not enough memory to expand the list.
2. The list tried to grow beyond its maximal capacity.
3. A non-existent element of the list was referenced. (i.e. the list index was out of bounds)
4. An attempt was made to add a duplicate entry to a `TStringList` (280) when `TStringList.AllowDuplicates` (280) is `False`.

2.21 EThread

Description

Thread error exception.

2.22 EThreadDestroyCalled

2.23 EWriteError

Description

If an error occurs when writing to a stream, a `EWriteError` exception is raised. Possible causes for this are:

1. The stream doesn't allow writing.
2. An error occurred when writing a property to a stream.

2.24 IStringsAdapter

Description

Is not yet supported in Free Pascal.

2.25 TAbstractObjectReader

Description

The Free Pascal streaming mechanism, while compatible with Delphi's mechanism, differs from it in the sense that the streaming mechanism uses a driver class when streaming components. The

`TAbstractObjectReader` class is the base driver class for reading property values from streams. It consists entirely of abstract methods, which must be implemented by descendent classes.

Different streaming mechanisms can be implemented by making a descendent from `TAbstractObjectReader`. The `TBinaryObjectReader` ([193](#)) class is such a descendent class, which streams data in binary (Delphi compatible) format.

All methods described in this class, mustbe implemented by descendent classes.

Method overview

Page	Method	Description
176	<code>BeginComponent</code>	Marks the reading of a new component.
176	<code>BeginProperty</code>	Marks the reading of a property value.
176	<code>BeginRootComponent</code>	Starts the reading of the root component.
175	<code>NextValue</code>	Returns the type of the next value in the stream.
176	<code>ReadBinary</code>	Read binary data from the stream.
177	<code>ReadDate</code>	Read a date value from the stream.
177	<code>ReadFloat</code>	Read a float value from the stream.
178	<code>ReadIdent</code>	Read an identifier from the stream.
179	<code>ReadInt16</code>	Read a 16-bit integer from the stream.
179	<code>ReadInt32</code>	Read a 32-bit integer from the stream.
179	<code>ReadInt64</code>	Read a 64-bit integer from the stream.
178	<code>ReadInt8</code>	Read an 8-bit integer from the stream.
180	<code>ReadSet</code>	Reads a set from the stream.
177	<code>ReadSingle</code>	Read a single (real-type) value from the stream.
180	<code>ReadStr</code>	Read a shortstring from the stream
180	<code>ReadString</code>	Read a string of type <code>StringType</code> from the stream.
175	<code>ReadValue</code>	Reads the type of the next value.
181	<code>SkipComponent</code>	Skip till the end of the component.
181	<code>SkipValue</code>	Skip the current value.

`TAbstractObjectReader.NextValue`

Synopsis: Returns the type of the next value in the stream.

Declaration: `function NextValue : TValueType; Virtual; Abstract`

Visibility: `public`

Description: This function should return the type of the next value in the stream, but should not read it, i.e. the stream position should not be altered by this method. This is used to 'peek' in the stream what value is next.

See also: `TAbstractObjectReader.ReadValue` ([175](#))

`TAbstractObjectReader.ReadValue`

Synopsis: Reads the type of the next value.

Declaration: `function ReadValue : TValueType; Virtual; Abstract`

Visibility: `public`

Description: This function returns the type of the next value in the stream and reads it. i.e. after the call to this method, the stream is positioned to read the value of the type returned by this function.

See also: `TAbstractObjectReader.ReadValue` ([175](#))

TAbstractObjectReader.BeginRootComponent

Synopsis: Starts the reading of the root component.

Declaration: `procedure BeginRootComponent; Virtual; Abstract`

Visibility: `public`

Description: This function can be used to initialize the driver class for reading a component. It is called once at the beginning of the read process, and is immediately followed by a call to `BeginComponent` ([176](#)).

See also: `TAbstractObjectReader.BeginComponent` ([176](#))

TAbstractObjectReader.BeginComponent

Synopsis: Marks the reading of a new component.

Declaration: `procedure BeginComponent(var Flags: TFileFlags; var AChildPos: Integer;
var CompClassName: String; var CompName: String)
; Virtual; Abstract`

Visibility: `public`

Description: This method is called when the streaming process wants to start reading a new component.

Descendent classes should override this method to read the start of a component new component definition and return the needed arguments. `Flags` should be filled with any flags that were found at the component definition, as well as `AChildPos`. The `CompClassName` should be filled with the class name of the streamed component, and the `CompName` argument should be filled with the name of the component.

See also: `TAbstractObjectReader.BeginRootComponent` ([176](#)), `TAbstractObjectReader.BeginProperty` ([176](#))

TAbstractObjectReader.BeginProperty

Synopsis: Marks the reading of a property value.

Declaration: `function BeginProperty : String; Virtual; Abstract`

Visibility: `public`

Description: `BeginProperty` is called by the streaming system when it wants to read a new property. The return value of the function is the name of the property which can be read from the stream.

See also: `TAbstractObjectReader.BeginComponent` ([176](#))

TAbstractObjectReader.ReadBinary

Synopsis: Read binary data from the stream.

Declaration: `procedure ReadBinary(const DestData: TMemoryStream); Virtual; Abstract`

Visibility: `public`

Description: `ReadBinary` is called when binary data should be read from the stream (i.e. after `ReadValue` (175) returned a valuetype of `vaBinary`). The data should be stored in the `DestData` memory stream by descendent classes.

See also: `TAbstractObjectReader.ReadFloat` (177), `TAbstractObjectReader.ReadDate` (177), `TAbstractObjectReader.ReadSingle` (177), `TAbstractObjectReader.ReadIdent` (178), `TAbstractObjectReader.ReadInt8` (178), `TAbstractObjectReader.ReadInt16` (179), `TAbstractObjectReader.ReadInt32` (179), `TAbstractObjectReader.ReadInt64` (179), `TabstractObjectReader.ReadSet` (180), `TabstractObjectReader.ReadStr` (180), `TabstractObjectReader.ReadString` (180)

TAbstractObjectReader.ReadFloat

Synopsis: Read a float value from the stream.

Declaration: `function ReadFloat : Extended; Virtual; Abstract`

Visibility: public

Description: `ReadFloat` is called by the streaming system when it wants to read a float from the stream (i.e. after `ReadValue` (175) returned a valuetype of `vaExtended`). The return value should be the value of the float.

See also: `TAbstractObjectReader.ReadFloat` (177), `TAbstractObjectReader.ReadDate` (177), `TAbstractObjectReader.ReadSingle` (177), `TAbstractObjectReader.ReadIdent` (178), `TAbstractObjectReader.ReadInt8` (178), `TAbstractObjectReader.ReadInt16` (179), `TAbstractObjectReader.ReadInt32` (179), `TAbstractObjectReader.ReadInt64` (179), `TabstractObjectReader.ReadSet` (180), `TabstractObjectReader.ReadStr` (180), `TabstractObjectReader.ReadString` (180)

TAbstractObjectReader.ReadSingle

Synopsis: Read a single (real-type) value from the stream.

Declaration: `function ReadSingle : Single; Virtual; Abstract`

Visibility: public

Description: `ReadSingle` is called by the streaming system when it wants to read a single-type float from the stream (i.e. after `ReadValue` (175) returned a valuetype of `vaSingle`). The return value should be the value of the float.

See also: `TAbstractObjectReader.ReadFloat` (177), `TAbstractObjectReader.ReadDate` (177), `TAbstractObjectReader.ReadSingle` (177), `TAbstractObjectReader.ReadIdent` (178), `TAbstractObjectReader.ReadInt8` (178), `TAbstractObjectReader.ReadInt16` (179), `TAbstractObjectReader.ReadInt32` (179), `TAbstractObjectReader.ReadInt64` (179), `TabstractObjectReader.ReadSet` (180), `TabstractObjectReader.ReadStr` (180), `TabstractObjectReader.ReadString` (180)

TAbstractObjectReader.ReadDate

Synopsis: Read a date value from the stream.

Declaration: `function ReadDate : TDateTime; Virtual; Abstract`

Visibility: public

Description: `ReadDate` is called by the streaming system when it wants to read a date/time value from the stream (i.e. after `ReadValue` (175) returned a valuetype of `vaDate`). The return value should be the date/time value. (This value can be stored as a float, since `TDateTime` is nothing but a float.)

See also: `TAbstractObjectReader.ReadFloat` (177), `TAbstractObjectReader.ReadSingle` (177), `TAbstractObjectReader.ReadIdent` (178), `TAbstractObjectReader.ReadInt8` (178), `TAbstractObjectReader.ReadInt16` (179), `TAbstractObjectReader.ReadInt32` (179), `TAbstractObjectReader.ReadInt64` (179), `TAbstractObjectReader.ReadSet` (180), `TAbstractObjectReader.ReadStr` (180), `TAbstractObjectReader.ReadString` (180)

TAbstractObjectReader.ReadIdent

Synopsis: Read an identifier from the stream.

Declaration: `function ReadIdent(ValueType: TValueType) : String; Virtual; Abstract`

Visibility: public

Description: `ReadIdent` is called by the streaming system if it expects to read an identifier of type `ValueType` from the stream after a call to `ReadValue` (175) returned `vaIdent`. The identifier should be returned as a string. Note that in some cases the identifier does not actually have to be in the stream;

Table 2.12:

ValueType	Expected value
<code>vaIdent</code>	Read from stream.
<code>vaNil</code>	'Nil'. This does not have to be read from the stream.
<code>vaFalse</code>	'False'. This does not have to be read from the stream.
<code>vaTrue</code>	'True'. This does not have to be read from the stream.
<code>vaNull</code>	'Null'. This does not have to be read from the stream.

See also: `TAbstractObjectReader.ReadFloat` (177), `TAbstractObjectReader.ReadDate` (177), `TAbstractObjectReader.ReadSingle` (177), `TAbstractObjectReader.ReadInt8` (178), `TAbstractObjectReader.ReadInt16` (179), `TAbstractObjectReader.ReadInt32` (179), `TAbstractObjectReader.ReadInt64` (179), `TAbstractObjectReader.ReadSet` (180), `TAbstractObjectReader.ReadStr` (180), `TAbstractObjectReader.ReadString` (180)

TAbstractObjectReader.ReadInt8

Synopsis: Read an 8-bit integer from the stream.

Declaration: `function ReadInt8 : ShortInt; Virtual; Abstract`

Visibility: public

Description: `ReadInt8` is called by the streaming process if it expects to read an integer value with a size of 8 bits (1 byte) from the stream (i.e. after `ReadValue` (175) returned a `valuetype` of `vaInt8`). The return value is the value if the integer. Note that the size of the value in the stream does not actually have to be 1 byte.

See also: `TAbstractObjectReader.ReadFloat` (177), `TAbstractObjectReader.ReadDate` (177), `TAbstractObjectReader.ReadSingle` (177), `TAbstractObjectReader.ReadIdent` (178), `TAbstractObjectReader.ReadInt16` (179), `TAbstractObjectReader.ReadInt32` (179), `TAbstractObjectReader.ReadInt64` (179), `TAbstractObjectReader.ReadSet` (180), `TAbstractObjectReader.ReadStr` (180), `TAbstractObjectReader.ReadString` (180)

TAbstractObjectReader.ReadInt16

Synopsis: Read a 16-bit integer from the stream.

Declaration: `function ReadInt16 : SmallInt; Virtual; Abstract`

Visibility: `public`

Description: `ReadInt16` is called by the streaming process if it expects to read an integer value with a size of 16 bits (2 bytes) from the stream (i.e. after `ReadValue` (175) returned a valuetype of `vaInt16`). The return value is the value if the integer. Note that the size of the value in the stream does not actually have to be 2 bytes.

See also: `TAbstractObjectReader.ReadFloat` (177), `TAbstractObjectReader.ReadDate` (177), `TAbstractObjectReader.ReadSingle` (177), `TAbstractObjectReader.ReadIdent` (178), `TAbstractObjectReader.ReadInt8` (178), `TAbstractObjectReader.ReadInt32` (179), `TAbstractObjectReader.ReadInt64` (179), `TabstractObjectReader.ReadSet` (180), `TabstractObjectReader.ReadStr` (180), `TabstractObjectReader.ReadString` (180)

TAbstractObjectReader.ReadInt32

Synopsis: Read a 32-bit integer from the stream.

Declaration: `function ReadInt32 : LongInt; Virtual; Abstract`

Visibility: `public`

Description: `ReadInt32` is called by the streaming process if it expects to read an integer value with a size of 32 bits (4 bytes) from the stream (i.e. after `ReadValue` (175) returned a valuetype of `vaInt32`). The return value is the value of the integer. Note that the size of the value in the stream does not actually have to be 4 bytes.

See also: `TAbstractObjectReader.ReadFloat` (177), `TAbstractObjectReader.ReadDate` (177), `TAbstractObjectReader.ReadSingle` (177), `TAbstractObjectReader.ReadIdent` (178), `TAbstractObjectReader.ReadInt8` (178), `TAbstractObjectReader.ReadInt16` (179), `TAbstractObjectReader.ReadInt64` (179), `TabstractObjectReader.ReadSet` (180), `TabstractObjectReader.ReadStr` (180), `TabstractObjectReader.ReadString` (180)

TAbstractObjectReader.ReadInt64

Synopsis: Read a 64-bit integer from the stream.

Declaration: `function ReadInt64 : Int64; Virtual; Abstract`

Visibility: `public`

Description: `ReadInt64` is called by the streaming process if it expects to read an `int64` value with a size of 64 bits (8 bytes) from the stream (i.e. after `ReadValue` (175) returned a valuetype of `vaInt64`). The return value is the value if the integer. Note that the size of the value in the stream does not actually have to be 8 bytes.

See also: `TAbstractObjectReader.ReadFloat` (177), `TAbstractObjectReader.ReadDate` (177), `TAbstractObjectReader.ReadSingle` (177), `TAbstractObjectReader.ReadIdent` (178), `TAbstractObjectReader.ReadInt8` (178), `TAbstractObjectReader.ReadInt16` (179), `TAbstractObjectReader.ReadInt32` (179), `TabstractObjectReader.ReadSet` (180), `TabstractObjectReader.ReadStr` (180), `TabstractObjectReader.ReadString` (180)

TAbstractObjectReader.ReadSet

Synopsis: Reads a set from the stream.

Declaration: `function ReadSet(EnumType: Pointer) : Integer; Virtual; Abstract`

Visibility: `public`

Description: This method is called by the streaming system if it expects to read a set from the stream (i.e. after `ReadValue` (175) returned a valuetype of `vaSet`). The return value is the contents of the set, encoded in a bitmask the following way:

For each (enumerated) value in the set, the bit corresponding to the ordinal value of the enumerated value should be set. i.e. as `1 shl ord(value)`.

See also: `TAbstractObjectReader.ReadFloat` (177), `TAbstractObjectReader.ReadDate` (177), `TAbstractObjectReader.ReadSingle` (177), `TAbstractObjectReader.ReadIdent` (178), `TAbstractObjectReader.ReadInt8` (178), `TAbstractObjectReader.ReadInt16` (179), `TAbstractObjectReader.ReadInt32` (179), `TAbstractObjectReader.ReadInt64` (179), `TabstractObjectReader.ReadStr` (180), `TabstractObjectReader.ReadString` (180)

TAbstractObjectReader.ReadStr

Synopsis: Read a shortstring from the stream

Declaration: `function ReadStr : String; Virtual; Abstract`

Visibility: `public`

Description: `ReadStr` is called by the streaming system if it expects to read a shortstring from the stream (i.e. after `ReadValue` (175) returned a valuetype of `vaLString`, `vaWstring` or `vaString`). The return value is the string.

See also: `TAbstractObjectReader.ReadFloat` (177), `TAbstractObjectReader.ReadDate` (177), `TAbstractObjectReader.ReadSingle` (177), `TAbstractObjectReader.ReadIdent` (178), `TAbstractObjectReader.ReadInt8` (178), `TAbstractObjectReader.ReadInt16` (179), `TAbstractObjectReader.ReadInt32` (179), `TAbstractObjectReader.ReadInt64` (179), `TabstractObjectReader.ReadSet` (180), `TabstractObjectReader.ReadString` (180)

TAbstractObjectReader.ReadString

Synopsis: Read a string of type `StringType` from the stream.

Declaration: `function ReadString(StringType: TValueType) : String; Virtual; Abstract`

Visibility: `public`

Description: `ReadStr` is called by the streaming system if it expects to read a string from the stream (i.e. after `ReadValue` (175) returned a valuetype of `vaLString`, `vaWstring` or `vaString`). The return value is the string.

See also: `TAbstractObjectReader.ReadFloat` (177), `TAbstractObjectReader.ReadDate` (177), `TAbstractObjectReader.ReadSingle` (177), `TAbstractObjectReader.ReadIdent` (178), `TAbstractObjectReader.ReadInt8` (178), `TAbstractObjectReader.ReadInt16` (179), `TAbstractObjectReader.ReadInt32` (179), `TAbstractObjectReader.ReadInt64` (179), `TabstractObjectReader.ReadSet` (180), `TabstractObjectReader.ReadStr` (180)

TAbstractObjectReader.SkipComponent

Synopsis: Skip till the end of the component.

Declaration: `procedure SkipComponent(SkipComponentInfos: Boolean); Virtual
; Abstract`

Visibility: public

Description: This method is used to skip the entire declaration of a component in the stream. Each descendent of `TAbstractObjectReader` should implement this in a way which is optimal for the implemented stream format.

See also: `TAbstractObjectReader.BeginComponent` (176), `TAbstractObjectReader.SkipValue` (181)

TAbstractObjectReader.SkipValue

Synopsis: Skip the current value.

Declaration: `procedure SkipValue; Virtual; Abstract`

Visibility: public

Description: `SkipValue` should be used when skipping a value in the stream; The method should determine the type of the value which should be skipped by itself, if this is necessary.

See also: `TAbstractObjectReader.SkipComponent` (181)

2.26 TAbstractObjectWriter**Description**

Abstract driver class for writing component data.

Method overview

Page	Method	Description
182	<code>BeginCollection</code>	Start writing a collection.
182	<code>BeginComponent</code>	Start writing a component
182	<code>BeginList</code>	Start writing a list.
182	<code>BeginProperty</code>	Start writing a property
182	<code>EndList</code>	Mark the end of a list.
182	<code>EndProperty</code>	Marks the end of writing of a property.
183	<code>WriteBinary</code>	Writes binary data to the stream.
183	<code>WriteBoolean</code>	Writes a boolean value to the stream.
183	<code>WriteDate</code>	Writes a date type to the stream.
183	<code>WriteFloat</code>	Writes a float value to the stream.
183	<code>WriteIdent</code>	Writes an identifier to the stream.
184	<code>WriteInteger</code>	Writes an integer value to the stream
184	<code>WriteMethodName</code>	Writes a methodname to the stream.
184	<code>WriteSet</code>	Writes a set value to the stream.
183	<code>WriteSingle</code>	Writes a single-type real value to the stream.
184	<code>WriteString</code>	Writes a string value to the stream.

TAbstractObjectWriter.BeginCollection

Synopsis: Start writing a collection.

Declaration: `procedure BeginCollection; Virtual; Abstract`

Visibility: public

Description: Start writing a collection.

TAbstractObjectWriter.BeginComponent

Synopsis: Start writing a component

Declaration: `procedure BeginComponent(Component: TComponent; Flags: TFileFlags;
ChildPos: Integer); Virtual; Abstract`

Visibility: public

Description: Start writing a component

TAbstractObjectWriter.BeginList

Synopsis: Start writing a list.

Declaration: `procedure BeginList; Virtual; Abstract`

Visibility: public

Description: Start writing a list.

TAbstractObjectWriter.EndList

Synopsis: Mark the end of a list.

Declaration: `procedure EndList; Virtual; Abstract`

Visibility: public

Description: Mark the end of a list.

TAbstractObjectWriter.BeginProperty

Synopsis: Start writing a property

Declaration: `procedure BeginProperty(const PropName: String); Virtual; Abstract`

Visibility: public

Description: Start writing a property

TAbstractObjectWriter.EndProperty

Synopsis: Marks the end of writing of a property.

Declaration: `procedure EndProperty; Virtual; Abstract`

Visibility: public

Description: Marks the end of writing of a property.

TAbstractObjectWriter.WriteBinary

Synopsis: Writes binary data to the stream.

Declaration: `procedure WriteBinary(const Buffer; Count: LongInt); Virtual; Abstract`

Visibility: public

Description: Writes binary data to the stream.

TAbstractObjectWriter.WriteBoolean

Synopsis: Writes a boolean value to the stream.

Declaration: `procedure WriteBoolean(Value: Boolean); Virtual; Abstract`

Visibility: public

Description: Writes a boolean value to the stream.

TAbstractObjectWriter.WriteFloat

Synopsis: Writes a float value to the stream.

Declaration: `procedure WriteFloat(const Value: Extended); Virtual; Abstract`

Visibility: public

Description: Writes a float value to the stream.

TAbstractObjectWriter.WriteSingle

Synopsis: Writes a single-type real value to the stream.

Declaration: `procedure WriteSingle(const Value: Single); Virtual; Abstract`

Visibility: public

Description: Writes a single-type real value to the stream.

TAbstractObjectWriter.WriteDate

Synopsis: Writes a date type to the stream.

Declaration: `procedure WriteDate(const Value: TDateTime); Virtual; Abstract`

Visibility: public

Description: Writes a date type to the stream.

TAbstractObjectWriter.WriteIdent

Synopsis: Writes an identifier to the stream.

Declaration: `procedure WriteIdent(const Ident: String); Virtual; Abstract`

Visibility: public

Description: Writes an identifier to the stream.

TAbstractObjectWriter.WriteInteger

Synopsis: Writes an integer value to the stream

Declaration: `procedure WriteInteger(Value: Int64); Virtual; Abstract`

Visibility: `public`

Description: Writes an integer value to the stream

TAbstractObjectWriter.WriteMethodName

Synopsis: Writes a methodname to the stream.

Declaration: `procedure WriteMethodName(const Name: String); Virtual; Abstract`

Visibility: `public`

Description: Writes a methodname to the stream.

TAbstractObjectWriter.WriteSet

Synopsis: Writes a set value to the stream.

Declaration: `procedure WriteSet(Value: LongInt; SetType: Pointer); Virtual; Abstract`

Visibility: `public`

Description: Writes a set value to the stream.

TAbstractObjectWriter.WriteString

Synopsis: Writes a string value to the stream.

Declaration: `procedure WriteString(const Value: String); Virtual; Abstract`

Visibility: `public`

Description: Writes a string value to the stream.

2.27 TBasicAction

Description

TBasicAction implements a basic action class from which all actions are derived. It introduces all basic methods of an action, and implements functionality to maintain a list of clients, i.e. components that are connected with this action.

Do not create instances of TBasicAction. Instead, create a descendent class and create an instance of this class instead.

Method overview

Page	Method	Description
185	Change	Calls the OnChange (188) handler.
185	Create	Creates a new instance of a TBasicAction (184) class.
186	Destroy	Destroys the action.
187	Execute	Triggers the OnExecute (188) event
187	ExecuteTarget	Executes the action on the Target object
186	HandlesTarget	Determines whether Target can be handled by this action
187	RegisterChanges	Registers a new client with the action.
185	SetOnExecute	Assigns an OnExecute (188) event handler
187	UnRegisterChanges	Unregisters a client from the list of clients
188	Update	Triggers the OnUpdate (189) event
186	UpdateTarget	Notify client controls when the action updates itself.

Property overview

Page	Property	Access	Description
188	ActionComponent	rw	Returns the component that initiated the action.
188	OnChange	rw	Occurs when one of the action's properties changes.
188	OnExecute	rw	Event triggered when the action executes.
189	OnUpdate	rw	Event triggered when the application is idle.

TBasicAction.Change

Synopsis: Calls the OnChange ([188](#)) handler.

Declaration: `procedure Change; Virtual`

Visibility: `protected`

Description: Change calls the OnChange ([188](#)) handler if one is assigned.

Application programmers should not call Change directly. It is called automatically if a property of an action component changes.

Descendent classes of TBasicAction should call explicitly call Change if one of their properties that affect client controls changes its value.

TBasicAction.SetOnExecute

Synopsis: Assigns an OnExecute ([188](#)) event handler

Declaration: `procedure SetOnExecute(Value: TNotifyEvent); Virtual`

Visibility: `protected`

Description: SetOnExecute sets the OnExecute ([188](#)) handler of the component. It also propagates this event to all client controls, and finally triggers the OnChange ([188](#)) event.

See also: TBasicAction.OnExecute ([188](#)), TBasicAction.OnChange ([188](#))

TBasicAction.Create

Synopsis: Creates a new instance of a TBasicAction ([184](#)) class.

Declaration: `constructor Create(AOwner: TComponent); Override`

Visibility: public

Description: `Create` calls the inherited constructor, and then initializes the list of clients controls (or action lists) by adding the `AClient` argument to the list of client controls.

Under normal circumstances it should not be necessary to create a `TBasicAction` descendent manually, actions are created in an IDE.

See also: `TBasicAction.Destroy` ([186](#)), `TBasicAction.AssignClient` ([184](#))

TBasicAction.Destroy

Synopsis: Destroys the action.

Declaration: `destructor Destroy; Override`

Visibility: public

Description: `Destroy` cleans up the list of client controls and then calls the inherited destructor.

An application programmer should not call `Destroy` directly; Instead `Free` should be called, if it needs to be called at all. Normally the controlling class (e.g. a `TActionList`) will destroy the action.

TBasicAction.HandlesTarget

Synopsis: Determines whether `Target` can be handled by this action

Declaration: `function HandlesTarget(Target: TObject) : Boolean; Virtual`

Visibility: public

Description: `HandlesTarget` returns `True` if `Target` is a valid client for this action and if so, if it is in a suitable state to execute the action. An application programmer should never need to call `HandlesTarget` directly, it will be called by the action itself when needed.

In `TBasicAction` this method is empty; descendent classes should override this method to implement appropriate checks.

See also: `TBasicAction.UpdateTarget` ([186](#)), `TBasicAction.ExecuteTarget` ([187](#))

TBasicAction.UpdateTarget

Synopsis: Notify client controls when the action updates itself.

Declaration: `procedure UpdateTarget(Target: TObject); Virtual`

Visibility: public

Description: `UpdateTarget` should update the client control specified by `Target` when the action updates itself. In `TBasicAction`, the implementation of `UpdateTarget` is empty. Descendent classes should override and implement `UpdateTarget` to actually update the `Target` object.

An application programmer should never need to call `HandlesTarget` directly, it will be called by the action itself when needed.

See also: `TBasicAction.HandlesTarget` ([186](#)), `TBasicAction.ExecuteTarget` ([187](#))

TBasicAction.ExecuteTarget

Synopsis: Executes the action on the Target object

Declaration: `procedure ExecuteTarget(Target: TObject); Virtual`

Visibility: public

Description: `ExecuteTarget` performs the action on the Target object. In `TBasicAction` this method does nothing. Descendent classes should implement the action to be performed. For instance an action to post data in a dataset could call the `Post` method of the dataset.

An application programmer should never call `ExecuteTarget` directly.

See also: `TBasicAction.HandlesTarget` ([186](#)), `TBasicAction.ExecuteTarget` ([187](#)), `TBasicAction.Execute` ([187](#))

TBasicAction.Execute

Synopsis: Triggers the `OnExecute` ([188](#)) event

Declaration: `function Execute : Boolean; Dynamic`

Visibility: public

Description: `Execute` triggers the `OnExecute` event, if one is assigned. It returns `True` if the event handler was called, `False` otherwise.

TBasicAction.RegisterChanges

Synopsis: Registers a new client with the action.

Declaration: `procedure RegisterChanges(Value: TBasicActionLink)`

Visibility: public

Description: `RegisterChanges` adds `Value` to the list of clients.

See also: `TBasicAction.UnregisterChanges` ([187](#))

TBasicAction.UnRegisterChanges

Synopsis: Unregisters a client from the list of clients

Declaration: `procedure UnRegisterChanges(Value: TBasicActionLink)`

Visibility: public

Description: `UnregisterChanges` removes `Value` from the list of clients. This is called for instance when the action is destroyed, or when the client is assigned a new action.

See also: `TBasicAction.UnregisterChanges` ([187](#)), `TBasicAction.Destroy` ([186](#))

TBasicAction.Update

Synopsis: Triggers the OnUpdate ([189](#)) event

Declaration: `function Update : Boolean; Virtual`

Visibility: `public`

Description: `Update` triggers the `OnUpdate` event, if one is assigned. It returns `True` if the event was triggered, or `False` if no event was assigned.

Application programmers should never run `Update` directly. The `Update` method is called automatically by the action mechanism; Normally this is in the Idle time of an application. An application programmer should assign the `OnUpdate` ([189](#)) event, and perform any checks in that handler.

See also: `TBasicAction.OnUpdate` ([189](#)), `TBasicAction.Execute` ([187](#)), `TBasicAction.UpdateTarget` ([186](#))

TBasicAction.OnChange

Synopsis: Occurs when one of the action's properties changes.

Declaration: `Property OnChange : TNotifyEvent`

Visibility: `protected`

Access: `Read,Write`

Description: `OnChange` is the event that is triggered when one of the action's properties changes. This event should be used by client controls or descendent classes to respond to these changes in the properties of the action.

Application programmers should never use the `OnChange` event directly.

TBasicAction.ActionComponent

Synopsis: Returns the component that initiated the action.

Declaration: `Property ActionComponent : TComponent`

Visibility: `public`

Access: `Read,Write`

Description: `ActionComponent` is set to the component that caused the action to execute, e.g. a toolbutton or a menu item. The property is set just before the action executes, and is reset to nil after the action was executed.

See also: `TBasicAction.Execute` ([187](#)), `TBasicAction.OnExecute` ([188](#))

TBasicAction.OnExecute

Synopsis: Event triggered when the action executes.

Declaration: `Property OnExecute : TNotifyEvent`

Visibility: `public`

Access: `Read,Write`

Description: `OnExecute` is the event triggered when the action is activated (executed). The event is triggered e.g. when the user clicks e.g. on a menu item or a button associated to the action. The application programmer should provide a `OnExecute` event handler to execute whatever code is necessary when the button is pressed or the menu item is chosen.

Note that assigning an `OnExecute` handler will result in the `Execute` (187) method returning a `True` value. Predefined actions (such as dataset actions) will check the result of `Execute` and will not perform their normal task if the `OnExecute` handler was called.

See also: `TBasicAction.Execute` (187), `TBasicAction.OnUpdate` (189)

TBasicAction.OnUpdate

Synopsis: Event triggered when the application is idle.

Declaration: Property `OnUpdate` : `TNotifyEvent`

Visibility: public

Access: Read,Write

Description: `OnUpdate` is the event triggered when the application is idle, and the action is being updated. The `OnUpdate` event can be used to set the state of the action, for instance disable it if the action cannot be executed at this point in time.

See also: `TBasicAction.Update` (188), `TBasicAction.OnExecute` (188)

2.28 TBasicActionLink

Description

`TBasicActionLink` links an Action to its clients. With each client for an action, a `TBasicActionLink` class is instantiated to handle the communication between the action and the client. It passes events between the action and its clients, and thus presents the action with a uniform interface to the clients.

An application programmer should never use a `TBasicActionLink` instance directly; They are created automatically when an action is associated with a component. Component programmers should create specialized descendents of `TBasicActionLink` which communicate changes in the action to the component.

Method overview

Page	Method	Description
190	<code>AssignClient</code>	Assigns a control (client) to the action link.
190	<code>Change</code>	Executed whenever the Action is changed.
191	<code>Create</code>	Creates a new instance of the <code>TBasicActionLink</code> class
191	<code>Destroy</code>	Destroys the <code>TBasicActionLink</code> instance.
191	<code>Execute</code>	Calls the action's <code>Execute</code> method.
190	<code>IsOnExecuteLinked</code>	Returns whether the client has it's <code>OnExecute</code> property linked.
190	<code>SetAction</code>	Sets the action with which the actionlink is associated.
191	<code>SetOnExecute</code>	Assigns the <code>OnExecute</code> (188) handler to the client
192	<code>Update</code>	Calls the action's <code>Update</code> method

Property overview

Page	Property	Access	Description
192	Action	rw	The action to which the link was assigned.
192	OnChange	rw	Event handler triggered when the action's properties change

TBasicActionLink.AssignClient

Synopsis: Assigns a control (client) to the action link.

Declaration: `procedure AssignClient(AClient: TObject); Virtual`

Visibility: protected

Description: `AssignClient` assigns a control to the actionlink and hence to the action. Descendent classes can override `AssignClient` to check whether the new client is a suitable client for this action.

See also: `TBasicActionLink.Action` ([192](#))

TBasicActionLink.Change

Synopsis: Executed whenever the Action is changed.

Declaration: `procedure Change; Virtual`

Visibility: protected

Description: `Change` is executed whenever the action changes. It executes the `OnChange` ([192](#)) handler, if one is assigned.

Component programmers may decide to override the `Change` procedure in descendent classes to perform additional actions when the properties of the action changes.

See also: `TBasicActionLink.OnChange` ([192](#)), `TBasicAction.Change` ([185](#))

TBasicActionLink.IsOnExecuteLinked

Synopsis: Returns whether the client has it's `OnExecute` property linked.

Declaration: `function IsOnExecuteLinked : Boolean; Virtual`

Visibility: protected

Description: `IsOnExecuteLinked` always returns true in `TBasicActionLink`. Descendent classes can override this method to provide a different result.

TBasicActionLink.SetAction

Synopsis: Sets the action with which the actionlink is associated.

Declaration: `procedure SetAction(Value: TBasicAction); Virtual`

Visibility: protected

Description: `SetAction` is the write handler for the `Action` ([192](#)) property. It sets the `Action` property to it's new value, after unregistering itself with the old action, if there was one.

See also: `TBasicActionLink.Action` ([192](#)), `TBasicAction` ([184](#))

TBasicActionLink.SetOnExecute

Synopsis: Assigns the OnExecute ([188](#)) handler to the client

Declaration: `procedure SetOnExecute(Value: TNotifyEvent); Virtual`

Visibility: `protected`

Description: `SetOnExecute` must be overridden by descendent classes to pass the OnExecute handler of the associated action to the client control. It will attach the OnExecute handler to whatever handler is appropriate for the client control.

See also: `TBasicAction.OnExecute` ([188](#)), `TBasicAction` ([184](#))

TBasicActionLink.Create

Synopsis: Creates a new instance of the `TBasicActionLink` class

Declaration: `constructor Create(AClient: TObject); Virtual`

Visibility: `public`

Description: `Create` creates a new instance of a `TBasicActionLink` and assigns `AClient` as the client of the link.

Application programmers should never instantiate `TBasicActionLink` classes directly. An instance is created automatically when an action is assigned to a control (client).

Component programmers can override the create constructor to initialize further properties.

See also: `TBasicActionLink.Destroy` ([191](#))

TBasicActionLink.Destroy

Synopsis: Destroys the `TBasicActionLink` instance.

Declaration: `destructor Destroy; Override`

Visibility: `public`

Description: `Destroy` unregisters the `TBasicActionLink` with the action, and then calls the inherited destructor.

Application programmers should never call `Destroy` directly. If a link should be destroyed at all, the `Free` method should be called instead.

See also: `TBasicActionLink.Create` ([191](#))

TBasicActionLink.Execute

Synopsis: Calls the action's `Execute` method.

Declaration: `function Execute(AComponent: TComponent) : Boolean; Virtual`

Visibility: `public`

Description: `Execute` sets the `ActionComponent` (188) property of the associated `Action` (192) to `AComponent` and then calls the `Action`'s `execute` (187) method. After the action has executed, the `ActionComponent` property is cleared again.

The return value of the function is the return value of the `Action`'s `execute` method.

Application programmers should never call `Execute` directly. This method will be called automatically when the associated control is activated. (e.g. a button is clicked on)

Component programmers should call `Execute` whenever the action should be activated.

See also: `TBasicActionLink.Action` (192), `TBasicAction.ActionComponent` (188), `TBasicAction.Execute` (187), `TBasicAction.onExecute` (188)

TBasicActionLink.Update

Synopsis: Calls the action's `Update` method

Declaration: `function Update : Boolean; Virtual`

Visibility: `public`

Description: `Update` calls the associated `Action`'s `Update` (188) method.

Component programmers can override the `Update` method to provide additional processing when the `Update` method occurs.

TBasicActionLink.Action

Synopsis: The action to which the link was assigned.

Declaration: `Property Action : TBasicAction`

Visibility: `public`

Access: `Read,Write`

Description: `Action` represents the `Action` (184) which was assigned to the client. Setting this property will unregister the client at the old action (if one existed) and registers the client at the new action.

See also: `TBasicAction` (184)

TBasicActionLink.OnChange

Synopsis: Event handler triggered when the action's properties change

Declaration: `Property OnChange : TNotifyEvent`

Visibility: `public`

Access: `Read,Write`

Description: `OnChange` is the event triggered when the action's properties change.

Application programmers should never need to assign this event. Component programmers can assign this event to have a client control reflect any changes in an `Action`'s properties.

See also: `TBasicActionLink.Change` (190), `TBasicAction.Change` (185)

2.29 TBinaryObjectReader

Description

The `TBinaryObjectReader` class reads component data stored in binary form in a file. For this, it overrides or implements all abstract methods from `TAbstractObjectReader` (174). No new functionality is added by this class, it is a driver class for the streaming system.

Method overview

Page	Method	Description
194	<code>BeginComponent</code>	
194	<code>BeginProperty</code>	
194	<code>BeginRootComponent</code>	
193	<code>Create</code>	Creates a new binary data reader instance.
193	<code>Destroy</code>	Destroys the binary data reader.
194	<code>NextValue</code>	
194	<code>ReadBinary</code>	
195	<code>ReadDate</code>	
194	<code>ReadFloat</code>	
195	<code>ReadIdent</code>	
195	<code>ReadInt16</code>	
195	<code>ReadInt32</code>	
195	<code>ReadInt64</code>	
195	<code>ReadInt8</code>	
195	<code>ReadSet</code>	
195	<code>ReadSingle</code>	
195	<code>ReadStr</code>	
196	<code>ReadString</code>	
194	<code>ReadValue</code>	
196	<code>SkipComponent</code>	
196	<code>SkipValue</code>	

TBinaryObjectReader.Create

Synopsis: Creates a new binary data reader instance.

Declaration: constructor `Create(Stream: TStream; BufSize: Integer)`

Visibility: public

Description: `Create` instantiates a new binary component data reader. The `Stream` stream is the stream from which data will be read. The `BufSize` argument is the size of the internal buffer that will be used by the reader. This can be used to optimize the reading process.

See also: `TAbstractObjectReader` (174)

TBinaryObjectReader.Destroy

Synopsis: Destroys the binary data reader.

Declaration: destructor `Destroy; Override`

Visibility: public

Description: Destroy frees the buffer allocated when the instance was created. It also positions the stream on the last used position in the stream (the buffering may cause the reader to read more bytes than were actually used.)

See also: `TBinaryObjectReader.Create` ([193](#))

TBinaryObjectReader.NextValue

Declaration: `function NextValue : TValueType; Override`

Visibility: public

TBinaryObjectReader.ReadValue

Declaration: `function ReadValue : TValueType; Override`

Visibility: public

TBinaryObjectReader.BeginRootComponent

Declaration: `procedure BeginRootComponent; Override`

Visibility: public

TBinaryObjectReader.BeginComponent

Declaration: `procedure BeginComponent(var Flags: TFileFlags; var AChildPos: Integer;
var CompClassName: String; var CompName: String)
; Override`

Visibility: public

TBinaryObjectReader.BeginProperty

Declaration: `function BeginProperty : String; Override`

Visibility: public

TBinaryObjectReader.ReadBinary

Declaration: `procedure ReadBinary(const DestData: TMemoryStream); Override`

Visibility: public

TBinaryObjectReader.ReadFloat

Declaration: `function ReadFloat : Extended; Override`

Visibility: public

TBinaryObjectReader.ReadSingle

Declaration: function ReadSingle : Single; Override

Visibility: public

TBinaryObjectReader.ReadDate

Declaration: function ReadDate : TDateTime; Override

Visibility: public

TBinaryObjectReader.ReadIdent

Declaration: function ReadIdent(ValueType: TValueType) : String; Override

Visibility: public

TBinaryObjectReader.ReadInt8

Declaration: function ReadInt8 : ShortInt; Override

Visibility: public

TBinaryObjectReader.ReadInt16

Declaration: function ReadInt16 : SmallInt; Override

Visibility: public

TBinaryObjectReader.ReadInt32

Declaration: function ReadInt32 : LongInt; Override

Visibility: public

TBinaryObjectReader.ReadInt64

Declaration: function ReadInt64 : Int64; Override

Visibility: public

TBinaryObjectReader.ReadSet

Declaration: function ReadSet(EnumType: Pointer) : Integer; Override

Visibility: public

TBinaryObjectReader.ReadStr

Declaration: function ReadStr : String; Override

Visibility: public

TBinaryObjectReader.ReadString

Declaration: `function ReadString(StringType: TValueType) : String; Override`

Visibility: `public`

TBinaryObjectReader.SkipComponent

Declaration: `procedure SkipComponent(SkipComponentInfos: Boolean); Override`

Visibility: `public`

TBinaryObjectReader.SkipValue

Declaration: `procedure SkipValue; Override`

Visibility: `public`

2.30 TBinaryObjectWriter**Description**

Driver class which stores component data in binary form.

Method overview

Page	Method	Description
197	<code>BeginCollection</code>	Start writing a collection.
197	<code>BeginComponent</code>	Start writing a component
197	<code>BeginList</code>	Start writing a list.
197	<code>BeginProperty</code>	Start writing a property
196	<code>Create</code>	Creates a new instance of a binary object writer.
197	<code>Destroy</code>	Destroys an instance of the binary object writer.
197	<code>EndList</code>	Mark the end of a list.
198	<code>EndProperty</code>	Marks the end of writing of a property.
198	<code>WriteBinary</code>	Writes binary data to the stream.
198	<code>WriteBoolean</code>	Writes a boolean value to the stream.
198	<code>WriteDate</code>	Writes a date type to the stream.
198	<code>WriteFloat</code>	Writes a float value to the stream.
198	<code>WriteIdent</code>	Writes an identifier to the stream.
199	<code>WriteInteger</code>	Writes an integer value to the stream.
199	<code>WriteMethodName</code>	Writes a methodname to the stream.
199	<code>WriteSet</code>	Writes a set value to the stream.
198	<code>WriteSingle</code>	Writes a single-type real value to the stream.
199	<code>WriteString</code>	Writes a string value to the stream.

TBinaryObjectWriter.Create

Synopsis: Creates a new instance of a binary object writer.

Declaration: `constructor Create(Stream: TStream; BufSize: Integer)`

Visibility: `public`

Description: Creates a new instance of a binary object writer.

TBinaryObjectWriter.Destroy

Synopsis: Destroys an instance of the binary object writer.

Declaration: `destructor Destroy; Override`

Visibility: `public`

Description: Destroys an instance of the binary object writer.

TBinaryObjectWriter.BeginCollection

Synopsis: Start writing a collection.

Declaration: `procedure BeginCollection; Override`

Visibility: `public`

TBinaryObjectWriter.BeginComponent

Synopsis: Start writing a component

Declaration: `procedure BeginComponent(Component: TComponent; Flags: TFileFlags;
ChildPos: Integer); Override`

Visibility: `public`

TBinaryObjectWriter.BeginList

Synopsis: Start writing a list.

Declaration: `procedure BeginList; Override`

Visibility: `public`

TBinaryObjectWriter.EndList

Synopsis: Mark the end of a list.

Declaration: `procedure EndList; Override`

Visibility: `public`

TBinaryObjectWriter.BeginProperty

Synopsis: Start writing a property

Declaration: `procedure BeginProperty(const PropName: String); Override`

Visibility: `public`

TBinaryObjectWriter.EndProperty

Synopsis: Marks the end of writing of a property.

Declaration: `procedure EndProperty; Override`

Visibility: `public`

TBinaryObjectWriter.WriteBinary

Synopsis: Writes binary data to the stream.

Declaration: `procedure WriteBinary(const Buffer; Count: LongInt); Override`

Visibility: `public`

TBinaryObjectWriter.WriteBoolean

Synopsis: Writes a boolean value to the stream.

Declaration: `procedure WriteBoolean(Value: Boolean); Override`

Visibility: `public`

TBinaryObjectWriter.WriteFloat

Synopsis: Writes a float value to the stream.

Declaration: `procedure WriteFloat(const Value: Extended); Override`

Visibility: `public`

TBinaryObjectWriter.WriteSingle

Synopsis: Writes a single-type real value to the stream.

Declaration: `procedure WriteSingle(const Value: Single); Override`

Visibility: `public`

TBinaryObjectWriter.WriteDate

Synopsis: Writes a date type to the stream.

Declaration: `procedure WriteDate(const Value: TDateTime); Override`

Visibility: `public`

TBinaryObjectWriter.WriteIdent

Synopsis: Writes an identifier to the stream.

Declaration: `procedure WriteIdent(const Ident: String); Override`

Visibility: `public`

TBinaryObjectWriter.WriteInteger

Synopsis: Writes an integer value to the stream.

Declaration: `procedure WriteInteger(Value: Int64); Override`

Visibility: `public`

TBinaryObjectWriter.WriteMethodName

Synopsis: Writes a methodname to the stream.

Declaration: `procedure WriteMethodName(const Name: String); Override`

Visibility: `public`

TBinaryObjectWriter.WriteSet

Synopsis: Writes a set value to the stream.

Declaration: `procedure WriteSet(Value: LongInt; SetType: Pointer); Override`

Visibility: `public`

TBinaryObjectWriter.WriteString

Synopsis: Writes a string value to the stream.

Declaration: `procedure WriteString(const Value: String); Override`

Visibility: `public`

2.31 TBits

Description

`TBits` can be used to store collections of bits in an indexed array. This is especially useful for storing collections of booleans: Normally the size of a boolean is the size of the smallest enumerated type, i.e. 1 byte. Since a bit can take 2 values it can be used to store a boolean as well. Since `TBits` can store 8 bits in a byte, it takes 8 times less space to store an array of booleans in a `TBits` class then it would take to store them in a conventional array.

`TBits` introduces methods to store and retrieve bit values, apply masks, and search for bits.

Method overview

Page	Method	Description
202	AndBits	Performs an and operation on the bits.
201	Clear	Clears a particular bit.
201	Clearall	Clears all bits in the array.
200	Create	Creates a new bits collection.
200	Destroy	Destroys a bit collection
203	Equals	Determines whether the bits of 2 arrays are equal.
204	FindFirstBit	Find first bit with a particular value
204	FindNextBit	Searches the next bit with a particular value.
205	FindPrevBit	Searches the previous bit with a particular value.
203	Get	Retrieve the value of a particular bit
201	GetFSize	Returns the number of records used to store the bits.
203	Grow	Expands the bits array to the requested size.
203	NotBits	Performs a not operation on the bits.
205	OpenBit	Returns the position of the first bit that is set to False.
202	OrBits	Performs an or operation on the bits.
204	SetIndex	Sets the start position for FindNextBit (204) and FindPrevBit (205)
201	SetOn	Turn a particular bit on.
202	XorBits	Performs a xor operation on the bits.

Property overview

Page	Property	Access	Description
205	Bits	rw	Access to all bits in the array.
206	Size	rw	Current size of the array of bits.

TBits.Create

Synopsis: Creates a new bits collection.

Declaration: `constructor Create(TheSize: LongInt); Virtual`

Visibility: `public`

Description: `Create` creates a new bit collection with initial size `TheSize`. The size of the collection can be changed later on.

All bits are initially set to zero.

See also: `TBits.Destroy` ([200](#))

TBits.Destroy

Synopsis: Destroys a bit collection

Declaration: `destructor Destroy; Override`

Visibility: `public`

Description: `Destroy` destroys a previously created bit collection and releases all memory used to store the bit collection.

`Destroy` should never be called directly, `Free` should be used instead.

Errors: None.

See also: `TBits.Create` ([200](#))

TBits.GetFSize

Synopsis: Returns the number of records used to store the bits.

Declaration: `function GetFSize : LongInt`

Visibility: `public`

Description: `GetFSize` returns the number of records used to store the current number of bits.

Errors: None.

See also: `TBits.Size` ([206](#))

TBits.SetOn

Synopsis: Turn a particular bit on.

Declaration: `procedure SetOn(Bit: LongInt)`

Visibility: `public`

Description: `SetOn` turns on the bit at position `bit`, i.e. sets it to 1. If `bit` is at a position bigger than the current size, the collection is expanded to the required size using `Grow` ([203](#)).

Errors: If `bit` is larger than the maximum allowed bits array size or is negative, an `EBitsError` ([171](#)) exception is raised.

See also: `TBits.Bits` ([205](#)), `TBits.clear` ([201](#))

TBits.Clear

Synopsis: Clears a particular bit.

Declaration: `procedure Clear(Bit: LongInt)`

Visibility: `public`

Description: `Clear` clears the bit at position `bit`. If the array `bit` is at a position bigger than the current size, the collection is expanded to the required size using `Grow` ([203](#)).

Errors: If `bit` is larger than the maximum allowed bits array size or is negative, an `EBitsError` ([171](#)) exception is raised.

See also: `TBits.Bits` ([205](#)), `TBits.clear` ([201](#))

TBits.Clearall

Synopsis: Clears all bits in the array.

Declaration: `procedure Clearall`

Visibility: `public`

Description: `ClearAll` clears all bits in the array, i.e. sets them to zero. `ClearAll` works faster than clearing all individual bits, since it uses the packed nature of the bits.

Errors: None.

See also: `TBits.Bits` ([205](#)), `TBits.clear` ([201](#))

TBits.AndBits

Synopsis: Performs an and operation on the bits.

Declaration: `procedure AndBits(BitSet: TBits)`

Visibility: `public`

Description: `andbits` performs an and operation on the bits in the array with the bits of array `BitSet`. If `BitSet` contains less bits than the current array, then all bits which have no counterpart in `BitSet` are cleared.

Errors: None.

See also: `TBits.clearall` ([201](#)), `TBits.orbits` ([202](#)), `TBits.xorbits` ([202](#)), `TBits.notbits` ([203](#))

TBits.OrBits

Synopsis: Performs an or operation on the bits.

Declaration: `procedure OrBits(BitSet: TBits)`

Visibility: `public`

Description: `andbits` performs an or operation on the bits in the array with the bits of array `BitSet`.

If `BitSet` contains less bits than the current array, then all bits which have no counterpart in `BitSet` are left untouched.

If the current array contains less bits than `BitSet` then it is grown to the size of `BitSet` before the or operation is performed.

Errors: None.

See also: `TBits.clearall` ([201](#)), `TBits.andbits` ([202](#)), `TBits.xorbits` ([202](#)), `TBits.notbits` ([203](#))

TBits.XorBits

Synopsis: Performs a xor operation on the bits.

Declaration: `procedure XorBits(BitSet: TBits)`

Visibility: `public`

Description: `XorBits` performs a xor operation on the bits in the array with the bits of array `BitSet`.

If `BitSet` contains less bits than the current array, then all bits which have no counterpart in `BitSet` are left untouched.

If the current array contains less bits than `BitSet` then it is grown to the size of `BitSet` before the xor operation is performed.

Errors: None.

See also: `TBits.clearall` ([201](#)), `TBits.andbits` ([202](#)), `TBits.orbits` ([202](#)), `TBits.notbits` ([203](#))

TBits.NotBits

Synopsis: Performs a not operation on the bits.

Declaration: `procedure NotBits(BitSet: TBits)`

Visibility: `public`

Description: `NotBits` performs a not operation on the bits in the array with the bits of array `BitSet`.

If `BitSet` contains less bits than the current array, then all bits which have no counterpart in `BitSet` are left untouched.

Errors: None.

See also: `TBits.clearall` (201), `TBits.andbits` (202), `TBits.orbits` (202), `TBits.xorbits` (202)

TBits.Get

Synopsis: Retrieve the value of a particular bit

Declaration: `function Get(Bit: LongInt) : Boolean`

Visibility: `public`

Description: `Get` returns `True` if the bit at position `bit` is set, or `False` if it is not set.

Errors: If `bit` is not a valid bit index then an `EBitsError` (171) exception is raised.

See also: `TBits.Bits` (205), `TBits.FindFirstBit` (204), `TBits.seton` (201)

TBits.Grow

Synopsis: Expands the bits array to the requested size.

Declaration: `procedure Grow(NBit: LongInt)`

Visibility: `public`

Description: `Grow` expands the bit array so it can at least contain `nbit` bits. If `nbit` is less than the current size, nothing happens.

Errors: If there is not enough memory to complete the operation, then an `EBitsError` (171) is raised.

See also: `TBits.Size` (206)

TBits.Equals

Synopsis: Determines whether the bits of 2 arrays are equal.

Declaration: `function Equals(BitSet: TBits) : Boolean`

Visibility: `public`

Description: `equals` returns `True` if all the bits in `BitSet` are the same as the ones in the current `BitSet`; if not, `False` is returned.

If the sizes of the two `BitSets` are different, the arrays are still reported equal when all the bits in the larger set, which are not present in the smaller set, are zero.

Errors: None.

See also: `TBits.clearall` (201), `TBits.andbits` (202), `TBits.orbits` (202), `TBits.xorbits` (202)

TBits.SetIndex

Synopsis: Sets the start position for FindNextBit (204) and FindPrevBit (205)

Declaration: `procedure SetIndex(Index: LongInt)`

Visibility: `public`

Description: `SetIndex` sets the search start position for `FindNextBit` (204) and `FindPrevBit` (205) to `Index`. This means that these calls will start searching from position `Index`.

This mechanism provides an alternative to `FindFirstBit` (204) which can also be used to position for the `FindNextBit` and `FindPrevBit` calls.

Errors: None.

See also: `TBits.FindNextBit` (204), `TBits.FindPrevBit` (205), `TBits.FindFirstBit` (204), `TBits.OpenBit` (205)

TBits.FindFirstBit

Synopsis: Find first bit with a particular value

Declaration: `function FindFirstBit(State: Boolean) : LongInt`

Visibility: `public`

Description: `FindFirstBit` searches for the first bit with value `State`. It returns the position of this bit, or `-1` if no such bit was found.

The search starts at position 0 in the array. If the first search returned a positive result, the found position is saved, and the `FindNextBit` (204) and `FindPrevBit` (205) will use this position to resume the search. To start a search from a certain position, the start position can be set with the `SetIndex` (204) instead.

Errors: None.

See also: `TBits.FindNextBit` (204), `TBits.FindPrevBit` (205), `TBits.OpenBit` (205), `TBits.SetIndex` (204)

TBits.FindNextBit

Synopsis: Searches the next bit with a particular value.

Declaration: `function FindNextBit : LongInt`

Visibility: `public`

Description: `FindNextBit` resumes a previously started search. It searches for the next bit with the value specified in the `FindFirstBit` (204). The search is done towards the end of the array and starts at the position last reported by one of the `Find` calls or at the position set with `SetIndex` (204).

If another bit with the same value is found, its position is returned. If no more bits with the same value are present in the array, `-1` is returned.

Errors: None.

See also: `TBits.FindFirstBit` (204), `TBits.FindPrevBit` (205), `TBits.OpenBit` (205), `TBits.SetIndex` (204)

TBits.FindPrevBit

Synopsis: Searches the previous bit with a particular value.

Declaration: `function FindPrevBit : LongInt`

Visibility: `public`

Description: `FindPrevBit` resumes a previously started search. It searches for the previous bit with the value specified in the `FindFirstBit` (204). The search is done towards the beginning of the array and starts at the position last reported by one of the `Find` calls or at the position set with `SetIndex` (204).

If another bit with the same value is found, its position is returned. If no more bits with the same value are present in the array, `-1` is returned.

Errors: None.

See also: `TBits.FindFirstBit` (204), `TBits.FindNextBit` (204), `TBits.OpenBit` (205), `TBits.SetIndex` (204)

TBits.OpenBit

Synopsis: Returns the position of the first bit that is set to `False`.

Declaration: `function OpenBit : LongInt`

Visibility: `public`

Description: `OpenBit` returns the position of the first bit whose value is `0` (`False`), or `-1` if no open bit was found. This call is equivalent to `FindFirstBit(False)`, except that it doesn't set the position for the next searches.

Errors: None.

See also: `TBits.FindFirstBit` (204), `TBits.FindPrevBit` (205), `TBits.FindFirstBit` (204), `TBits.SetIndex` (204)

TBits.Bits

Synopsis: Access to all bits in the array.

Declaration: `Property Bits[Bit: LongInt]: Boolean; default`

Visibility: `public`

Access: `Read, Write`

Description: `Bits` allows indexed access to all of the bits in the array. It gives `True` if the bit is `1`, `False` otherwise; Assigning to this property will set, respectively clear the bit.

Errors: If an index is specified which is out of the allowed range then an `EBitsError` (171) exception is raised.

See also: `TBits.Size` (206)

TBits.Size

Synopsis: Current size of the array of bits.

Declaration: Property Size : LongInt

Visibility: public

Access: Read,Write

Description: Size is the current size of the bit array. Setting this property will adjust the size; this is equivalent to calling Grow(Value-1)

Errors: If an invalid size (negative or too large) is specified, a EBitsError (171) exception is raised.

See also: TBits.Bits (205)

2.32 TCollection**Description**

TCollection implements functionality to manage a collection of named objects. Each of these objects needs to be a descendent of the TCollectionItem (213) class. Exactly which type of object is managed can be seen from the TCollection.ItemClass (212) property.

Normally, no TCollection is created directly. Instead, a descendent of TCollection and TCollectionItem (213) are created as a pair.

Method overview

Page	Method	Description
210	Add	Creates and adds a new item to the collection.
210	Assign	Assigns one collection to another.
210	BeginUpdate	Start an update batch.
208	Changed	Procedure called if an item is added to or removed from the collection.
211	Clear	Removes all items from the collection.
209	Create	Creates a new collection.
210	Destroy	Destroys the collection and frees all the objects it manages.
211	EndUpdate	Ends an update batch.
211	FindItemID	Searches for an Item in the collection, based on its TCollectionItem.ID (216) property.
207	GetAttr	Returns an attribute of the collection.
207	GetAttrCount	Returns the count of attributes associated with each item.
208	GetItem	Read handler for the TCollection.Items (212) property.
207	GetItemAttr	Returns the attributes of an item.
207	GetNamePath	Overrides TPersistent.GetNamePath (258) to return a proper pathname.
208	SetItem	Write handler for the TCollection.Items (212) property.
209	SetItemName	Virtual method to set the name of the specified item
209	SetPropName	Write handler for the TCollection.PropName (212) property
209	Update	Handler called when an item in the collection has changed.

Property overview

Page	Property	Access	Description
212	Count	r	Number of items in the collection.
212	ItemClass	r	Class pointer for each item in the collection.
212	Items	rw	Indexed array of items in the collection.
212	PropName	rw	Name of the property that this collection represents.

TCollection.GetAttrCount

Synopsis: Returns the count of attributes associated with each item.

Declaration: `function GetAttrCount : Integer; Dynamic`

Visibility: protected

Description: `GetAttrCount` returns 0 in the `TCollection` implementation. It can be used to determine the number of attributes associated with each collection item. Descendent objects should override this method to return the number of attributes.

This method is provided for compatibility with Delphi only and is not used in Free Pascal.

See also: `TCollection.GetAttr` ([207](#)), `TCollection.GetItemAttr` ([207](#))

TCollection.GetAttr

Synopsis: Returns an attribute of the collection.

Declaration: `function GetAttr(Index: Integer) : String; Dynamic`

Visibility: protected

Description: This method is provided for compatibility with Delphi only and is not used in Free Pascal.

See also: `TCollection.GetAttrCount` ([207](#)), `TCollection.GetItemAttr` ([207](#))

TCollection.GetItemAttr

Synopsis: Returns the attributes of an item.

Declaration: `function GetItemAttr(Index: Integer; ItemIndex: Integer) : String
; Dynamic`

Visibility: protected

Description: This method is provided for compatibility with Delphi only and is not used in Free Pascal.

See also: `TCollection.GetAttr` ([207](#)), `TCollection.GetAttrCount` ([207](#))

TCollection.GetNamePath

Synopsis: Overrides `TPersistent.GetNamePath` ([258](#)) to return a proper pathname.

Declaration: `function GetNamePath : String; Override`

Visibility: protected

Description: `GetNamePath` returns the name path for this collection. If the following conditions are satisfied:

1. There is an owner object.
2. The owner object returns a non-empty name path.
3. The `TCollection.Propname` (212) property is not empty

collection has an owner and the owning object has a name, then the function returns the owner name, followed by the propname. If one of the conditions is not satisfied, then the classname is returned.

See also: `TCollection.GetOwner` (206), `TCollection.Propname` (212)

TCollection.Changed

Synopsis: Procedure called if an item is added to or removed from the collection.

Declaration: `procedure Changed`

Visibility: `protected`

Description: `Changed` is called if a change takes place in the collection managed by the class. If the update count has reached zero, it calls `TCollection.Update` (209) with a nil argument.

See also: `TCollection.Update` (209), `TCollection.Add` (210), `TCollection.Clear` (211)

TCollection.GetItem

Synopsis: Read handler for the `TCollection.Items` (212) property.

Declaration: `function GetItem(Index: Integer) : TCollectionItem`

Visibility: `protected`

Description: `GetItem` is the read handler for the `TCollection.Items` (212) property. It returns the `Index`-th element from the list of objects.

Errors: If `Index` is outside the allowed range, then an `EListError` (172) exception is raised.

See also: `TCollection.Items` (212), `TCollection.Count` (212), `TCollection.SetItem` (208)

TCollection.SetItem

Synopsis: Write handler for the `TCollection.Items` (212) property.

Declaration: `procedure SetItem(Index: Integer; Value: TCollectionItem)`

Visibility: `protected`

Description: `SetItem` implements the write handler for the `TCollection.Items` (212) property. It assigns `Value` to the `Index`-th element in the array. For this to work properly, the `TPersistent.Assign` (257) method of the `Item` must work correctly.

Errors: If `Index` is outside the allowed range, then an `EListError` (172) exception is raised.

See also: `TCollection.Items` (212), `TCollection.Count` (212), `TCollection.GetItem` (208)

TCollection.SetItemName

Synopsis: Virtual method to set the name of the specified item

Declaration: `procedure SetItemName(Item: TCollectionItem); Virtual`

Visibility: protected

Description: Virtual method to set the name of the specified item

TCollection.SetPropName

Synopsis: Write handler for the TCollection.PropName (212) property

Declaration: `procedure SetPropName; Virtual`

Visibility: protected

Description: SetPropName must be overridden by descendent objects to set the TCollection.PropName (212) property to a suitable value. By default, SetPropName sets the PropName property to empty.

See also: TCollection.PropName (212)

TCollection.Update

Synopsis: Handler called when an item in the collection has changed.

Declaration: `procedure Update(Item: TCollectionItem); Virtual`

Visibility: protected

Description: Update is called in the following cases:

1. An item is added to or removed from the collection.
2. An item is moved in the list, i.e. its TCollectionItem.Index (216) property changes.
3. An item's TCollectionItem.DisplayName (217) property changes.

Descendent classes can override this method to perform additional actions when the collection changes. The Item parameter indicates the item that was changed. This can be Nil

See also: TCollection.Changed (208)

TCollection.Create

Synopsis: Creates a new collection.

Declaration: `constructor Create(AItemClass: TCollectionItemClass)`

Visibility: public

Description: Create instantiates a new instance of the TCollection class which will manage objects of class AItemClass. It creates the list used to hold all objects, and stores the AItemClass for the adding of new objects to the collection.

See also: TCollection.ItemClass (212), TCollection.Destroy (210)

TCollection.Destroy

Synopsis: Destroys the collection and frees all the objects it manages.

Declaration: `destructor Destroy; Override`

Visibility: `public`

Description: `Destroy` first clears the collection, and then frees all memory allocated to this instance.

Don't call `Destroy` directly, call `Free` instead.

See also: `TCollection.Create` (209)

TCollection.Add

Synopsis: Creates and adds a new item to the collection.

Declaration: `function Add : TCollectionItem`

Visibility: `public`

Description: `Add` instantiates a new item of class `TCollection.ItemClass` (212) and adds it to the list. The newly created object is returned.

See also: `TCollection.ItemClass` (212), `TCollection.Clear` (211)

TCollection.Assign

Synopsis: Assigns one collection to another.

Declaration: `procedure Assign(Source: TPersistent); Override`

Visibility: `public`

Description: `Assign` assigns the contents of one collection to another. It does this by clearing the items list, and adding as much elements as there are in the `Source` collection; it assigns to each created element the contents of it's counterpart in the `Source` element.

Two collections cannot be assigned to each other if instances of the `ItemClass` classes cannot be assigned to each other.

Errors: If the objects in the collections cannot be assigned to one another, then an `EConvertError` is raised.

See also: `TPersistent.Assign` (257), `TCollectionItem` (213)

TCollection.BeginUpdate

Synopsis: Start an update batch.

Declaration: `procedure BeginUpdate`

Visibility: `public`

Description: `BeginUpdate` is called at the beginning of a batch update. It raises the update count with 1.

Call `BeginUpdate` at the beginning of a series of operations that will change the state of the collection. This will avoid the call to `TCollection.Update` (209) for each operation. At the end of the operations, a corresponding call to `EndUpdate` must be made. It is best to do this in the context of a `Try ... finally` block:

```

With MyCollection Do
  try
    BeginUpdate;
    // Some Lengthy operations
  finally
    EndUpdate;
  end;

```

This insures that the number of calls to `BeginUpdate` always matches the number of calls to `TCollection.EndUpdate` (211), even in case of an exception.

See also: `TCollection.EndUpdate` (211), `TCollection.Changed` (208), `TCollection.Update` (209)

TCollection.Clear

Synopsis: Removes all items from the collection.

Declaration: `procedure Clear`

Visibility: `public`

Description: `Clear` will clear the collection, i.e. each item in the collection is destroyed and removed from memory. After a call to `Clear`, `Count` is zero.

See also: `TCollection.Add` (210), `TCollectionItem.Destroy` (215), `TCollection.Destroy` (210)

TCollection.EndUpdate

Synopsis: Ends an update batch.

Declaration: `procedure EndUpdate`

Visibility: `public`

Description: `EndUpdate` signals the end of a series of operations that change the state of the collection, possibly triggering an update event. It does this by decreasing the update count with 1 and calling `TCollection.Changed` (208) it should always be used in conjunction with `TCollection.BeginUpdate` (210), preferably in the `Finally` section of a `Try ... Finally` block.

See also: `TCollection.BeginUpdate` (210), `TCollection.Changed` (208), `TCollection.Update` (209)

TCollection.FindItemID

Synopsis: Searches for an Item in the collection, based on its `TCollectionItem.ID` (216) property.

Declaration: `function FindItemID(ID: Integer) : TCollectionItem`

Visibility: `public`

Description: `FindItemID` searches through the collection for the item that has a value of `ID` for its `TCollectionItem.ID` (216) property, and returns the found item. If no such item is found in the collection, `Nil` is returned.

The routine performs a linear search, so this can be slow on very large collections.

See also: `TCollection.Items` (212), `TCollectionItem.ID` (216)

TCollection.PropName

Synopsis: Name of the property that this collection represents.

Declaration: `Property PropName : String`

Visibility: `protected`

Access: `Read,Write`

Description: `PropName` indicates the name of the property that this collection is supposed to represent. By default, this is the empty string. Descendents can override this property to return the name of the property that is represented by this collection.

See also: `TCollection.SetPropName` (209), `TCollection.GetPropName` (206)

TCollection.Count

Synopsis: Number of items in the collection.

Declaration: `Property Count : Integer`

Visibility: `public`

Access: `Read`

Description: `Count` contains the number of items in the collection.

Remark: The items in the collection are identified by their `TCollectionItem.Index` (216) property, which is a zero-based index, meaning that it can take values between 0 and `Count`.

See also: `TCollectionItem.Index` (216), `TCollection.Items` (212)

TCollection.ItemClass

Synopsis: Class pointer for each item in the collection.

Declaration: `Property ItemClass : TCollectionItemClass`

Visibility: `public`

Access: `Read`

Description: `ItemClass` is the class pointer with which each new item in the collection is created. It is the value that was passed to the collection's constructor when it was created, and does not change during the lifetime of the collection.

See also: `TCollectionItem` (213), `TCollection.Items` (212)

TCollection.Items

Synopsis: Indexed array of items in the collection.

Declaration: `Property Items[Index: Integer]: TCollectionItem`

Visibility: `public`

Access: `Read,Write`

Description: `Items` provides indexed access to the items in the collection. Since the array is zero-based, `Index` should be an integer between 0 and `Count-1`.

It is possible to set or retrieve an element in the array. When setting an element of the array, the object that is assigned should be compatible with the class of the objects in the collection, as given by the `TCollection.ItemClass` (212) property.

Adding an element to the array can be done with the `TCollection.Add` (210) method. The array can be cleared with the `TCollection.Clear` (211) method. Removing an element of the array should be done by freeing that element.

See also: `TCollection.Count` (212), `TCollection.ItemClass` (212), `TCollection.Clear` (211), `TCollection.Add` (210)

2.33 TCollectionItem

Description

`TCollectionItem` and `TCollection` (206) form a pair of base classes that manage a collection of named objects. The `TCollectionItem` is the named object that is managed, it represents one item in the collection. An item in the collection is represented by two properties: `TCollectionItem.DisplayName` (217), `TCollection.Index` (206) and `TCollectionItem.ID` (216).

A `TCollectionItem` object is never created directly. To manage a set of named items, it is necessary to make a descendent of `TCollectionItem` to which needed properties and methods are added. This descendant can then be managed with a `TCollection` (206) class. The managing collection will create and destroy its items by itself, it should therefore never be necessary to create `TCollectionItem` descendents manually.

Method overview

Page	Method	Description
214	<code>Changed</code>	Method to notify the managing collection that the name or index of this item has changed.
215	<code>Create</code>	Creates a new instance of this collection item.
215	<code>Destroy</code>	Destroys this collection item.
214	<code>GetDisplayName</code>	Returns the <code>TCollectionItem.DisplayName</code> (217) of the collection-item
214	<code>GetNamePath</code>	Returns the namepath of this collection item.
214	<code>GetOwner</code>	Returns the managing collection.
215	<code>SetDisplayName</code>	Write method for the <code>TCollectionItem.DisplayName</code> (217) property
215	<code>SetIndex</code>	Write method for the <code>TCollectionItem.Index</code> (216) property.

Property overview

Page	Property	Access	Description
216	<code>Collection</code>	rw	Pointer to the collection managing this item.
217	<code>DisplayName</code>	rw	Name of the item, displayed in the object inspector.
216	<code>ID</code>	r	Initial index of this item.
216	<code>Index</code>	rw	Index of the item in its managing collection <code>TCollection.Items</code> (212) property.

TCollectionItem.Changed

Synopsis: Method to notify the managing collection that the name or index of this item has changed.

Declaration: `procedure Changed(AllItems: Boolean)`

Visibility: `protected`

Description: This method is called when the `TCollectionItem.DisplayName` (217) is set or when the `TCollectionItem.Index` (216) is changed.

See also: `TCollectionItem.Id` (216), `TCollectionItem.Index` (216), `TCollection.Update` (209)

TCollectionItem.GetNamePath

Synopsis: Returns the namepath of this collection item.

Declaration: `function GetNamePath : String; Override`

Visibility: `protected`

Description: `GetNamePath` overrides the `TPersistent.GetNamePath` (258) method to return the name of the managing collection and appends its `TCollectionItem.Index` (216) property.

See also: `TCollectionItem.Collection` (216), `TPersistent.GetNamePath` (258), `TCollectionItem.Index` (216)

TCollectionItem.GetOwner

Synopsis: Returns the managing collection.

Declaration: `function GetOwner : TPersistent; Override`

Visibility: `protected`

Description: `TCollectionItem` overrides `TPersistent.GetOwner` (257) to and returns the `TCollectionItem.Collection` (216) that manages it.

See also: `TPersistent.GetOwner` (257), `TCollectionItem.Collection` (216)

TCollectionItem.GetDisplayName

Synopsis: Returns the `TCollectionItem.DisplayName` (217) of the collectionitem

Declaration: `function GetDisplayName : String; Virtual`

Visibility: `protected`

Description: `GetDisplayName` returns the value of the `TCollectionItem.DisplayName` (217) property. By default, this is the classname of the actual `TCollectionItem` descendant.

Descendants of `TCollectionItem` can and should override this method to return a more meaningful value.

See also: `TCollectionItem.DisplayName` (217)

TCollectionItem.SetIndex

Synopsis: Write method for the TCollectionItem.Index (216) property.

Declaration: `procedure SetIndex(Value: Integer); Virtual`

Visibility: `protected`

Description: `SetIndex` implements the write handler for the TCollectionItem.Index (216) property. It requests the managing collection to move this item to the desired index value.

See also: TCollectionItem.Index (216)

TCollectionItem.SetDisplayName

Synopsis: Write method for the TCollectionItem.DisplayName (217) property

Declaration: `procedure SetDisplayName(const Value: String); Virtual`

Visibility: `protected`

Description: `SetDisplayName` is the write method for the TCollectionItem.DisplayName (217) property. It does nothing but notifying the managing collection that the displayname has changed. It does NOT store the actual Value.

Descendants of TCollectionItem should override this method to store the actual displayname if this is required.

See also: TCollectionItem.DisplayName (217)

TCollectionItem.Create

Synopsis: Creates a new instance of this collection item.

Declaration: `constructor Create(ACollection: TCollection); Virtual`

Visibility: `public`

Description: `Create` instantiates a new item in a TCollection (206). It is called by the TCollection.Add (210) function and should under normal circumstances never be called directly. called

See also: TCollectionItem.Destroy (215)

TCollectionItem.Destroy

Synopsis: Destroys this collection item.

Declaration: `destructor Destroy; Override`

Visibility: `public`

Description: `Destroy` removes the item from the managing collection and Destroys the item instance.

This is the only way to remove items from a collection;

See also: TCollectionItem.Create (215)

TCollectionItem.Collection

Synopsis: Pointer to the collection managing this item.

Declaration: Property `Collection` : `TCollection`

Visibility: public

Access: Read,Write

Description: `Collection` points to the collection managing this item. This property can be set to point to a new collection. If this is done, the old collection will be notified that the item should no longer be managed, and the new collection is notified that it should manage this item as well.

See also: `TCollection` ([206](#))

TCollectionItem.ID

Synopsis: Initial index of this item.

Declaration: Property `ID` : `Integer`

Visibility: public

Access: Read

Description: `ID` is the initial value of `TCollectionItem.Index` ([216](#)); it doesn't change after the index changes. It can be used to uniquely identify the item. The `ID` property doesn't change as items are added and removed from the collection.

While the `TCollectionItem.Index` ([216](#)) property forms a continuous series, `ID` does not. If items are removed from the collection, their `ID` is not used again, leaving gaps. Only when the collection is initially created, the `ID` and `Index` properties will be equal.

See also: `TCollection.Items` ([212](#)), `TCollectionItem.Index` ([216](#))

TCollectionItem.Index

Synopsis: Index of the item in its managing collection `TCollection.Items` ([212](#)) property.

Declaration: Property `Index` : `Integer`

Visibility: public

Access: Read,Write

Description: `Index` is the current index of the item in its managing collection's `TCollection.Items` ([212](#)) property. This property may change as items are added and removed from the collection.

The index of an item is zero-based, i.e. the first item has index zero. The last item has index `Count-1` where `Count` is the number of items in the collection.

The `Index` property of the items in a collection form a continuous series ranging from 0 to `Count-1`. The `TCollectionItem.ID` ([216](#)) property does not form a continuous series, but can also be used to identify an item.

See also: `TCollectionItem.ID` ([216](#)), `TCollection.Items` ([212](#))

TCollectionItem.DisplayName

Synopsis: Name of the item, displayed in the object inspector.

Declaration: `Property DisplayName : String`

Visibility: `public`

Access: `Read,Write`

Description: `DisplayName` contains the name of this item as shown in the object inspector. For `TCollectionItem` this returns always the class name of the managing collection, followed by the index of the item.

`TCollectionItem` does not implement any functionality to store the `DisplayName` property. The property can be set, but this will have no effect other than that the managing collection is notified of a change. The actual displayname will remain unchanged. To store the `DisplayName` property, `TCollectionItem` descendants should override the `TCollectionItem.SetDisplayName` (215) and `TCollectionItem.GetDisplayName` (214) to add storage functionality.

See also: `TCollectionItem.Index` (216), `TCollectionItem.ID` (216), `TCollectionItem.GetDisplayName` (214), `TCollectionItem.SetDisplayName` (215)

2.34 TComponent

Description

`TComponent` is the base class for any set of classes that needs owner-owned functionality, and which needs support for property streaming. All classes that should be handled by an IDE (Integrated Development Environment) must descend from `TComponent`, as it includes all support for streaming all its published properties.

Components can 'own' other components. `TComponent` introduces methods for enumerating the child components. It also allows to name the owned components with a unique name. Furthermore, functionality for sending notifications when a component is removed from the list or removed from memory altogether is also introduced in `TComponent`.

`TComponent` introduces a form of automatic memory management: When a component is destroyed, all its child components will be destroyed first.

Method overview

Page	Method	Description
225	BeforeDestruction	
219	ChangeName	Actually sets the component name.
225	Create	Creates a new instance of the component.
219	DefineProperties	Defines fake top,left properties for handling in the IDE.
225	Destroy	Destroys the instance of the component.
226	DestroyComponents	Destroy child components.
226	Destroying	Called when the component is being destroyed
226	ExecuteAction	
226	FindComponent	Finds and returns the named component in the owned components.
227	FreeNotification	Ask the component to notify called when it is being destroyed.
227	FreeOnRelease	Part of the <code>IVCLComObject</code> interface.
220	GetChildOwner	Returns the owner of any children.
220	GetChildParent	Returns the parent of any children.
219	GetChildren	Must be overridden by descendents to return all child components that must be streamed.
220	GetNamePath	Returns the name path of this component.
220	GetOwner	Returns the owner of this component.
227	GetParentComponent	Returns the parent component.
227	HasParent	Does the component have a parent ?
228	InsertComponent	Insert the given component in the list of owned components.
221	Loaded	Called when the component has finished loading.
221	Notification	Called by components that are freed and which received a <code>FreeNotification</code> .
221	ReadState	Read the component's state from a stream.
228	RemoveComponent	Remove the given component from the list of owned components.
227	RemoveFreeNotification	
228	SafeCallException	Part of the <code>IVCLComObject</code> Interface.
222	SetAncestor	Sets the <code>csAncestor</code> state of the component.
223	SetChildOrder	Determines the order in which children are streamed/created.
222	SetDesigning	Sets the <code>csDesigning</code> state of the component.
222	SetName	Write handler for Name (231) property.
223	SetParentComponent	Set the parent component.
228	UpdateAction	
223	Updated	Ends the <code>csUpdating</code> state.
224	UpdateRegistry	For compatibilty only.
223	Updating	Sets the state to <code>csUpdating</code>
224	ValidateContainer	??
224	ValidateInsert	Called when an insert must be validated.
224	ValidateRename	Called when a name change must be validated
225	WriteState	Writes the component to a stream.

Property overview

Page	Property	Access	Description
229	ComponentCount	r	Count of owned components
229	ComponentIndex	rw	Index of component in it's owner's list.
228	Components	r	Indexed list (zero-based) of all owned components.
229	ComponentState	r	Current component's state.
230	ComponentStyle	r	Current component's style.
230	DesignInfo	rw	Information for IDE designer.
231	Name	rws	Name of the component.
230	Owner	r	Owner of this component.
231	Tag	rw	Tag value of the component.
230	VCLComObject	rw	Not implemented.

TComponent.ChangeName

Synopsis: Actually sets the component name.

Declaration: `procedure ChangeName(const NewName: TComponentName)`

Visibility: protected

Description: `ChangeName` is called by the `SetName` ([222](#)) procedure when the component name is set and the name has been verified. It actually sets the name of the component to `NewName`, and can be used to bypass the name checks which are done when the `Name` ([231](#)) property is set.

Application programmers should never use `SetName` directly.

See also: `TComponent.SetName` ([222](#)), `TComponent.Name` ([231](#))

TComponent.DefineProperties

Synopsis: Defines fake top,left properties for handling in the IDE.

Declaration: `procedure DefineProperties(Filer: TFile); Override`

Visibility: protected

Description: `DefineProperties` overrides the standard `TPersistent.DefineProperties` ([256](#)) to store the top/left properties used to display an icon for a non-visual component in an IDE.

See also: `TPersistent.DefineProperties` ([256](#))

TComponent.GetChildren

Synopsis: Must be overridden by descendents to return all child components that must be streamed.

Declaration: `procedure GetChildren(Proc: TGetChildProc;Root: TComponent); Dynamic`

Visibility: protected

Description: `GetChildren` is called by the streaming system to determine which child components should be streamed as well when the component is being streamed. By default, no child components are streamed, i.e. the `TComponent` implementation is empty.

`TComponent` descendents should override this method. For each child that needs to be streamed, `Proc` should be called with as an argument the child component that must be streamed. The `Root` argument contains the root component relative to which all streaming is done.

See also: `TComponent.WriteState` ([225](#))

TComponent.GetChildOwner

Synopsis: Returns the owner of any children.

Declaration: `function GetChildOwner : TComponent; Dynamic`

Visibility: `protected`

Description: `GetChildOwner` returns the owner of the children that are read from the stream. If the method returns `Nil` (the default) this means that streamed child components are owned by the root component of the streaming process (usually a Form or Datamodule)

Application programmers should not call `GetChildOwner` directly, it is called by the streaming system when needed.

See also: `TComponent.WriteState` ([225](#)), `TComponent.ReadState` ([221](#)), `TComponent.Owner` ([230](#)), `TComponent.GetChildParent` ([220](#))

TComponent.GetChildParent

Synopsis: Returns the parent of any children.

Declaration: `function GetChildParent : TComponent; Dynamic`

Visibility: `protected`

Description: `GetChildParent` returns the parent component of the child components being streamed. The parent property is a visual property, which is not always meaningful. If there is no parent component, the owner of child components that are streamed is returned. If `Nil` is returned, then the root component of the streaming operation is assumed. The `TComponent` implementation of this method returns `Self`.

Application programmers should not call this method, it is called automatically by the streaming mechanism.

See also: `TComponent.GetChildOwner` ([220](#))

TComponent.GetNamePath

Synopsis: Returns the name path of this component.

Declaration: `function GetNamePath : String; Override`

Visibility: `protected`

Description: `GetNamePath` returns the name of the component as it will be shown in the object inspector.

`TComponent` overrides `GetNamePath` so it returns the `Name` ([231](#)) property of the component.

See also: `TComponent.Name` ([231](#)), `TPersistent.GetNamePath` ([258](#))

TComponent.GetOwner

Synopsis: Returns the owner of this component.

Declaration: `function GetOwner : TPersistent; Override`

Visibility: `protected`

Description: `GetOwner` returns the owner of this component as indicated by the `Owner` (230) property. The `GetOwner` call is introduced in `TPersistent` (255) and is used by the streaming system to determine the 'owner' of a component.

See also: `TPersistent.GetOwner` (257), `TComponent.Owner` (230)

TComponent.Loaded

Synopsis: Called when the component has finished loading.

Declaration: `procedure Loaded; Virtual`

Visibility: protected

Description: `Loaded` is called by the streaming system when a root component was completely read from a stream and all properties and references to other objects have been resolved by the streaming system. Descendents of `TComponent` should override this method to do some additional processing of properties after all published properties have been set from values obtained from the stream.

Application programmers should never call `Loaded` directly, this is done automatically by the streaming system.

See also: `TComponent.ReadState` (221), `TComponent.ComponentState` (229)

TComponent.Notification

Synopsis: Called by components that are freed and which received a `FreeNotification`.

Declaration: `procedure Notification(AComponent: TComponent; Operation: TOperation)
; Virtual`

Visibility: protected

Description: `Notification` is called whenever a child component is destroyed, inserted or removed from the list of owned component. Components that were requested to send a notification when they are freed ((with `FreeNotification` (227)) will also call `Notification` when they are freed.

The `AComponent` parameter specifies which component sends the notification, and `Operation` specifies whether the component is being inserted into or removed from the child component list, or whether it is being destroyed.

Descendents of `TComponent` can use `FreeNotification` (227) to request notification of the destruction of another object. By overriding the `Notification` method, they can do special processing (typically, set a reference to this component to `Nil`) when this component is destroyed. The `Notification` method is called quite often in the streaming process, so speed should be a consideration when overriding this method.

See also: `TOperation` (157), `TComponent.FreeNotification` (227)

TComponent.ReadState

Synopsis: Read the component's state from a stream.

Declaration: `procedure ReadState(Reader: TReader); Virtual`

Visibility: protected

Description: `ReadState` reads the component's state from a stream through the reader object `reader`. Values for all published properties of the component can be read from the stream. Normally there is no need to call `ReadState` directly. The streaming system calls `ReadState` itself.

The `TComponent` (217) implementation of `ReadState` simply calls `TReader.ReadData` (261). Descendent classes can, however, override `ReadState` to provide additional processing of stream data.

See also: `TComponent.WriteState` (225), `TStream.ReadComponent` (275), `TReader.ReadData` (261)

TComponent.SetAncestor

Synopsis: Sets the `csAncestor` state of the component.

Declaration: `procedure SetAncestor(Value: Boolean)`

Visibility: `protected`

Description: `SetAncestor` includes or excludes the `csAncestor` flag in the `ComponentState` (229) set property, depending on the boolean `Value`. The flag is set recursively for all owned components as well.

This is normally only done during the streaming system, and should not be called directly by an application programmer.

See also: `TComponent.ComponentState` (229)

TComponent.SetDesigning

Synopsis: Sets the `csDesigning` state of the component.

Declaration: `procedure SetDesigning(Value: Boolean)`

Visibility: `protected`

Description: `SetDesigning` includes or excludes the `csDesigning` flag in the `ComponentState` (229) set property, depending on the boolean `Value`. The flag is set recursively for all owned components as well.

This is normally only done during the streaming system, and should not be called directly by an application programmer.

TComponent.SetName

Synopsis: Write handler for `Name` (231) property.

Declaration: `procedure SetName(const NewName: TComponentName); Virtual`

Visibility: `protected`

Description: `SetName` is the write handler for the `Name` (231) property. It checks whether the desired name is valid (i.e. is a valid identifier) and is unique among the children of the owner component. If either conditions is not satisfied, an exception is raised.

See also: `TComponent.Name` (231), `TComponent.ValidateRename` (224)

TComponent.SetChildOrder

Synopsis: Determines the order in which children are streamed/created.

Declaration: `procedure SetChildOrder(Child: TComponent; Order: Integer); Dynamic`

Visibility: `protected`

Description: This method does nothing. It can be used to change the order in which child components are streamed and created. This can be used by descendent classes to optimize or correct the order in which child components are streamed.

See also: `TComponent.ReadState` ([221](#))

TComponent.SetParentComponent

Synopsis: Set the parent component.

Declaration: `procedure SetParentComponent(Value: TComponent); Dynamic`

Visibility: `protected`

Description: `SetParentComponent` does nothing, but is called by the streaming system to set the parent component of the current component. This method can be overridden by descendent components to set the parent component of the current component.

See also: `TComponent.Owner` ([230](#))

TComponent.Updating

Synopsis: Sets the state to `csUpdating`

Declaration: `procedure Updating; Dynamic`

Visibility: `protected`

Description: `Updating` includes `csUpdating` in the `ComponentState` ([229](#)) property of the component.

Normally, an application programmer should not call this method directly, it is called automatically by the streaming system.

See also: `TComponent.Updated` ([223](#)), `TComponent.ComponentState` ([229](#))

TComponent.Updated

Synopsis: Ends the `csUpdating` state.

Declaration: `procedure Updated; Dynamic`

Visibility: `protected`

Description: `Updated` excludes `csUpdating` from the `ComponentState` ([229](#)) property of the component.

Normally, an application programmer should not call this method directly, it is called automatically by the streaming system.

See also: `TComponent.Updating` ([223](#)), `TComponent.ComponentState` ([229](#))

TComponent.UpdateRegistry

Synopsis: For compatibility only.

Declaration: `procedure UpdateRegistry(Register: Boolean;const ClassID: String;
const ProgID: String); Dynamic`

Visibility: protected

Description: This method does nothing, and is provided for compatibility only.

TComponent.ValidateRename

Synopsis: Called when a name change must be validated

Declaration: `procedure ValidateRename(AComponent: TComponent;const CurName: String;
const NewName: String); Virtual`

Visibility: protected

Description: `ValidateRename` checks whether `NewName` is a valid replacement for `CurName` for component `AComponent`. Two owned components of a component can not have the same name. If a child component with the same name is found, then an exception is raised.

See also: `TComponent.SetName` ([222](#)), `TComponent.Name` ([231](#))

TComponent.ValidateContainer

Synopsis: ??

Declaration: `procedure ValidateContainer(AComponent: TComponent); Dynamic`

Visibility: protected

Description: `ValidateContainer` is provided for compatibility only. It doesn't do anything in Free Pascal.

TComponent.ValidateInsert

Synopsis: Called when an insert must be validated.

Declaration: `procedure ValidateInsert(AComponent: TComponent); Dynamic`

Visibility: protected

Description: `ValidateInsert` should be implemented by descendent components to see whether the `AComponent` component may be inserted in the list of owned components.

This procedure does nothing in the `TComponent` implementation, it should be overridden by descendant components.

See also: `TComponent.Insert` ([217](#))

TComponent.WriteState

Synopsis: Writes the component to a stream.

Declaration: `procedure WriteState(Writer: TWriter); Virtual`

Visibility: `public`

Description: `WriteState` writes the component's current state to a stream through the writer (311) object `writer`. Values for all published properties of the component can be written to the stream. Normally there is no need to call `WriteState` directly. The streaming system calls `WriteState` itself.

The `TComponent` (217) implementation of `WriteState` simply calls `TWriter.WriteData` (311). Descendent classes can, however, override `WriteState` to provide additional processing of stream data.

See also: `TComponent.ReadState` (221), `TStream.WriteComponent` (275), `TWriter.WriteData` (311)

TComponent.Create

Synopsis: Creates a new instance of the component.

Declaration: `constructor Create(AOwner: TComponent); Virtual`

Visibility: `public`

Description: `Create` creates a new instance of a `TComponent` class. If `AOwner` is not `Nil`, the new component attempts to insert itself in the list of owned components of the owner.

See also: `TComponent.Insert` (217), `TComponent.Owner` (230)

TComponent.BeforeDestruction

Declaration: `procedure BeforeDestruction; Override`

Visibility: `public`

TComponent.Destroy

Synopsis: Destroys the instance of the component.

Declaration: `destructor Destroy; Override`

Visibility: `public`

Description: `Destroy` sends a `opRemove` notification to all components in the free-notification list. After that, all owned components are destroyed by calling `DestroyComponents` (226) (and hence removed from the list of owned components). When this is done, the component removes itself from its owner's child component list. After that, the parent's destroy method is called.

See also: `TComponent.Notification` (221), `TComponent.Owner` (230), `TComponent.DestroyComponents` (226), `TComponent.Components` (228)

TComponent.DestroyComponents

Synopsis: Destroy child components.

Declaration: `procedure DestroyComponents`

Visibility: `public`

Description: `DestroyComponents` calls the destructor of all owned components, till no more components are left in the `Components` (228) array.

Calling the destructor of an owned component has as the effect that the component will remove itself from the list of owned components, if nothing has disrupted the sequence of destructors.

Errors: If an overridden 'destroy' method does not call it's inherited destructor or raises an exception, it's `TComponent.Destroy` (225) destructor will not be called, which may result in an endless loop.

See also: `TComponent.Destroy` (225), `TComponent.Components` (228)

TComponent.Destroying

Synopsis: Called when the component is being destroyed

Declaration: `procedure Destroying`

Visibility: `public`

Description: `Destroying` sets the `csDestroying` flag in the component's state (217) property, and does the same for all owned components.

It is not necessary to call `Destroying` directly, the destructor `Destroy` (225) does this automatically.

See also: `TComponent.State` (217), `TComponent.Destroy` (225)

TComponent.ExecuteAction

Declaration: `function ExecuteAction(Action: TBasicAction) : Boolean; Dynamic`

Visibility: `public`

TComponent.FindComponent

Synopsis: Finds and returns the named component in the owned components.

Declaration: `function FindComponent(const AName: String) : TComponent`

Visibility: `public`

Description: `FindComponent` searches the component with name `AName` in the list of owned components. If `AName` is empty, then `Nil` is returned.

See also: `TComponent.Components` (228), `TComponent.Name` (231)

TComponent.FreeNotification

Synopsis: Ask the component to notify called when it is being destroyed.

Declaration: `procedure FreeNotification(AComponent: TComponent)`

Visibility: `public`

Description: `FreeNotification` inserts `AComponent` in the `freenotification` list. When the component is destroyed, the `Notification` (221) method is called for all components in the `freenotification` list.

See also: `TComponent.Components` (228), `TComponent.Notification` (221)

TComponent.RemoveFreeNotification

Declaration: `procedure RemoveFreeNotification(AComponent: TComponent)`

Visibility: `public`

TComponent.FreeOnRelease

Synopsis: Part of the `IVCLComObject` interface.

Declaration: `procedure FreeOnRelease`

Visibility: `public`

Description: Provided for Delphi compatibility, but is not yet implemented.

TComponent.GetParentComponent

Synopsis: Returns the parent component.

Declaration: `function GetParentComponent : TComponent; Dynamic`

Visibility: `public`

Description: `GetParentComponent` can be implemented to return the parent component of this component. The implementation of this method in `TComponent` always returns `Nil`. Descendent classes must override this method to return the visual parent of the component.

See also: `TComponent.HasParent` (227), `TComponent.Owner` (230)

TComponent.HasParent

Synopsis: Does the component have a parent ?

Declaration: `function HasParent : Boolean; Dynamic`

Visibility: `public`

Description: `HasParent` can be implemented to return whether the parent of the component exists. The implementation of this method in `TComponent` always returns `False`, and should be overridden by descendent classes to return `True` when a parent is available. If `HasParent` returns `True`, then `GetParentComponent` (227) will return the parent component.

See also: `TComponent.HasParent` (227), `TComponent.Owner` (230)

TComponent.InsertComponent

Synopsis: Insert the given component in the list of owned components.

Declaration: `procedure InsertComponent(AComponent: TComponent)`

Visibility: `public`

Description: `InsertComponent` attempts to insert `AComponent` in the list with owned components. It first calls `ValidateComponent` (217) to see whether the component can be inserted. It then checks whether there are no name conflicts by calling `ValidateRename` (224). If neither of these checks have raised an exception the component is inserted, and notified of the insert.

See also: `TComponent.RemoveComponent` (228), `TComponent.Insert` (217), `TComponent.ValidateContainer` (224), `TComponent.ValidateRename` (224), `TComponent.Notification` (221)

TComponent.RemoveComponent

Synopsis: Remove the given component from the list of owned components.

Declaration: `procedure RemoveComponent(AComponent: TComponent)`

Visibility: `public`

Description: `RemoveComponent` will send an `opRemove` notification to `AComponent` and will then proceed to remove `AComponent` from the list of owned components.

See also: `TComponent.InsertComponent` (228), `TComponent.Remove` (217), `TComponent.ValidateRename` (224), `TComponent.Notification` (221)

TComponent.SafeCallException

Synopsis: Part of the `IVCLComObject` Interface.

Declaration: `function SafeCallException(ExceptObject: TObject; ExceptAddr: Pointer)
: Integer; Override`

Visibility: `public`

Description: Provided for Delphi compatibility, but not implemented.

TComponent.UpdateAction

Declaration: `function UpdateAction(Action: TBasicAction) : Boolean; Dynamic`

Visibility: `public`

TComponent.Components

Synopsis: Indexed list (zero-based) of all owned components.

Declaration: `Property Components[Index: Integer]: TComponent`

Visibility: `public`

Access: `Read`

Description: `Components` provides indexed access to the list of owned components. `Index` can range from 0 to `ComponentCount-1` (229).

See also: `TComponent.ComponentCount` (229), `TComponent.Owner` (230)

TComponent.ComponentCount

Synopsis: Count of owned components

Declaration: Property ComponentCount : Integer

Visibility: public

Access: Read

Description: ComponentCount returns the number of components that the current component owns. It can be used to determine the valid index range in the Component (228) array.

See also: TComponent.Components (228), TComponent.Owner (230)

TComponent.ComponentIndex

Synopsis: Index of component in it's owner's list.

Declaration: Property ComponentIndex : Integer

Visibility: public

Access: Read,Write

Description: ComponentIndex is the index of the current component in its owner's list of components. If the component has no owner, the value of this property is -1.

See also: TComponent.Components (228), TComponent.ComponentCount (229), TComponent.Owner (230)

TComponent.ComponentState

Synopsis: Current component's state.

Declaration: Property ComponentState : TComponentState

Visibility: public

Access: Read

Description: ComponentState indicates the current state of the component. It is a set of flags which indicate the various stages in the lifetime of a component. The following values can occur in this set:

Table 2.13: Component states

Flag	Meaning
csLoading	The component is being loaded from stream
csReading	Component properties are being read from stream.
csWriting	Component properties are being written to stream.
csDestroying	The component or one of it's owners is being destroyed.
csAncestor	The component is being streamed as part of a frame
csUpdating	The component is being updated
csFixups	References to other components are being resolved
csFreeNotification	The component has freenotifications.
csInline	The component is being loaded as part of a frame
csDesignInstance	? not used.

The component state is set by various actions such as reading it from stream, destroying it etc.

See also: [TComponent.SetAncestor \(222\)](#), [TComponent.SetDesigning \(222\)](#), [TComponent.SetInline \(217\)](#), [TComponent.SetDesignInstance \(217\)](#), [TComponent.Updating \(223\)](#), [TComponent.Updated \(223\)](#), [TComponent.Loaded \(221\)](#)

TComponent.ComponentStyle

Synopsis: Current component's style.

Declaration: `Property ComponentStyle : TComponentStyle`

Visibility: `public`

Access: `Read`

Description: Current component's style.

TComponent.DesignInfo

Synopsis: Information for IDE designer.

Declaration: `Property DesignInfo : LongInt`

Visibility: `public`

Access: `Read,Write`

Description: `DesignInformation` can be used by an IDE to store design information in the component. It should not be used by an application programmer.

See also: [TComponent.Tag \(231\)](#)

TComponent.Owner

Synopsis: Owner of this component.

Declaration: `Property Owner : TComponent`

Visibility: `public`

Access: `Read`

Description: `Owner` returns the owner of this component. The owner cannot be set except by explicitly inserting the component in another component's owned components list using that component's [InsertComponent \(228\)](#) method, or by removing the component from it's owner's owned component list using the [RemoveComponent \(228\)](#) method.

See also: [TComponent.Components \(228\)](#), [TComponent.InsertComponent \(228\)](#), [TComponent.RemoveComponent \(228\)](#)

TComponent.VCLComObject

Synopsis: Not implemented.

Declaration: `Property VCLComObject : Pointer`

Visibility: `public`

Access: `Read,Write`

Description: `VCLComObject` is not yet implemented in Free Pascal.

TComponent.Name

Synopsis: Name of the component.

Declaration: Property Name : TComponentName

Visibility: published

Access: Read,Write

Description: Name is the name of the component. This name should be a valid identifier, i.e. must start with a letter, and can contain only letters, numbers and the underscore character. When attempting to set the name of a component, the name will be checked for validity. Furthermore, when a component is owned by another component, the name must be either empty or must be unique among the child component names.

Errors: Attempting to set the name to an invalid value will result in an exception being raised.

See also: TComponent.ValidateRename ([224](#)), TComponent.Owner ([230](#))

TComponent.Tag

Synopsis: Tag value of the component.

Declaration: Property Tag : LongInt

Visibility: published

Access: Read,Write

Description: Tag can be used to store an integer value in the component. This value is streamed together with all other published properties. It can be used for instance to quickly identify a component in an event handler.

See also: TComponent.Name ([231](#))

2.35 TCustomMemoryStream**Description**

TCustomMemoryStream is the parent class for streams that stored their data in memory. It introduces all needed functions to handle reading from and navigating through the memory, and introduces a Memory ([234](#)) property which points to the memory area where the stream data is kept.

The only thing which TCustomMemoryStream does not do is obtain memory to store data when writing data or the writing of data. This functionality is implemented in descendent streams such as TMemoryStream ([248](#)). The reason for this approach is that this way it is possible to create e.g. read-only descendents of TCustomMemoryStream that point to a fixed part in memory which can be read from, but not written to.

Remark: Since TCustomMemoryStream is an abstract class, do not create instances of TMemoryStream directly. Instead, create instances of descendents such as TMemoryStream ([248](#)).

Method overview

Page	Method	Description
232	Read	Reads Count bytes from the stream into buffer.
233	SaveToFile	Writes the contents of the stream to a file.
233	SaveToStream	Writes the contents of the memory stream to another stream.
232	Seek	Sets a new position in the stream.
232	SetPointer	Sets the internal memory pointer and size of the memory block.

Property overview

Page	Property	Access	Description
234	Memory	r	Pointer to the data kept in the memory stream.

TCustomMemoryStream.SetPointer

Synopsis: Sets the internal memory pointer and size of the memory block.

Declaration: `procedure SetPointer(Ptr: Pointer; ASize: LongInt)`

Visibility: `protected`

Description: `SetPointer` updates the internal memory pointer and the size of the memory area pointed to.

Descendent memory streams should call this method whenever they set or reset the memory the stream should read from or write to.

See also: `TCustomMemoryStream.Memory` ([234](#)), `TStream.Size` ([280](#))

TCustomMemoryStream.Read

Synopsis: Reads Count bytes from the stream into buffer.

Declaration: `function Read(var Buffer; Count: LongInt) : LongInt; Override`

Visibility: `public`

Description: `Read` reads Count bytes from the stream into the memory pointed to by buffer. It returns the number of bytes actually read.

This method overrides the abstract `TStream.Read` ([272](#)) method of `TStream` ([271](#)). It will read as much bytes as are still available in the memory area pointer to by `Memory` ([234](#)). After the bytes are read, the internal stream position is updated.

See also: `TCustomMemoryStream.Memory` ([234](#)), `TStream.Read` ([272](#))

TCustomMemoryStream.Seek

Synopsis: Sets a new position in the stream.

Declaration: `function Seek(Offset: LongInt; Origin: Word) : LongInt; Override`

Visibility: `public`

Description: `Seek` overrides the abstract `TStream.Seek` ([273](#)) method. It simply updates the internal stream position, and returns the new position.

Errors: No checking is done whether the new position is still a valid position, i.e. whether the position is still within the range `0..Size`. Attempting a seek outside the valid memory range of the stream may result in an exception at the next read or write operation.

See also: `TStream.Position` (280), `TStream.Size` (280), `TCustomMemoryStream.Memory` (234)

TCustomMemoryStream.SaveToStream

Synopsis: Writes the contents of the memory stream to another stream.

Declaration: `procedure SaveToStream(Stream: TStream)`

Visibility: public

Description: `SaveToStream` writes the contents of the memory stream to `Stream`. The content of `Stream` is not cleared first. The current position of the memory stream is not changed by this action.

Remark: This method will work much faster than the use of the `TStream.CopyFrom` (274) method:

```
Seek(0, soFromBeginning);
Stream.CopyFrom(Self, Size);
```

because the `CopyFrom` method copies the contents in blocks, while `SaveToStream` writes the contents of the memory as one big block.

Errors: If an error occurs when writing to `Stream` an `EStreamError` (173) exception will be raised.

See also: `TCustomMemoryStream.SaveToFile` (233), `TStream.CopyFrom` (274)

TCustomMemoryStream.SaveToFile

Synopsis: Writes the contents of the stream to a file.

Declaration: `procedure SaveToFile(const FileName: String)`

Visibility: public

Description: `SaveToFile` writes the contents of the stream to a file with name `FileName`. It simply creates a filestream and writes the contents of the memorystream to this file stream using `TCustomMemoryStream.SaveToStream` (233).

Remark: This method will work much faster than the use of the `TStream.CopyFrom` (274) method:

```
Stream:=TFileStream.Create(fmCreate, FileName);
Seek(0, soFromBeginning);
Stream.CopyFrom(Self, Size);
```

because the `CopyFrom` method copies the contents in blocks, while `SaveToFile` writes the contents of the memory as one big block.

Errors: If an error occurs when creating or writing to the file, an `EStreamError` (173) exception may occur.

See also: `TCustomMemoryStream.SaveToStream` (233), `TFileStream` (238), `TStream.CopyFrom` (274)

TCustomMemoryStream.Memory

Synopsis: Pointer to the data kept in the memory stream.

Declaration: `Property Memory : Pointer`

Visibility: `public`

Access: `Read`

Description: `Memory` points to the memory area where stream keeps it's data. The property is read-only, so the pointer cannot be set this way.

Remark: Do not write to the memory pointed to by `Memory`, since the memory content may be read-only, and thus writing to it may cause errors.

See also: `TStream.Size` ([280](#))

2.36 TDataModule**Method overview**

Page	Method	Description
235	<code>AfterConstruction</code>	
235	<code>BeforeDestruction</code>	
235	<code>Create</code>	
235	<code>CreateNew</code>	
235	<code>DefineProperties</code>	
235	<code>Destroy</code>	
234	<code>DoCreate</code>	
234	<code>DoDestroy</code>	
235	<code>GetChildren</code>	
235	<code>HandleCreateException</code>	
235	<code>ReadState</code>	

Property overview

Page	Property	Access	Description
236	<code>DesignOffset</code>	<code>rw</code>	
236	<code>DesignSize</code>	<code>rw</code>	
236	<code>OldCreateOrder</code>	<code>rw</code>	
236	<code>OnCreate</code>	<code>rw</code>	
236	<code>OnDestroy</code>	<code>rw</code>	

TDataModule.DoCreate

Declaration: `procedure DoCreate; Virtual`

Visibility: `protected`

TDataModule.DoDestroy

Declaration: `procedure DoDestroy; Virtual`

Visibility: `protected`

TDataModule.DefineProperties

Declaration: procedure DefineProperties(Filer: TFile); Override

Visibility: protected

TDataModule.GetChildren

Declaration: procedure GetChildren(Proc: TGetChildProc; Root: TComponent); Override

Visibility: protected

TDataModule.HandleCreateException

Declaration: function HandleCreateException : Boolean; Virtual

Visibility: protected

TDataModule.ReadState

Declaration: procedure ReadState(Reader: TReader); Override

Visibility: protected

TDataModule.Create

Declaration: constructor Create(AOwner: TComponent); Override

Visibility: public

TDataModule.CreateNew

Declaration: constructor CreateNew(AOwner: TComponent)

constructor CreateNew(AOwner: TComponent; CreateMode: Integer); Virtual

Visibility: public

TDataModule.Destroy

Declaration: destructor Destroy; Override

Visibility: public

TDataModule.AfterConstruction

Declaration: procedure AfterConstruction; Override

Visibility: public

TDataModule.BeforeDestruction

Declaration: procedure BeforeDestruction; Override

Visibility: public

TDataModule.DesignOffset

Declaration: Property DesignOffset : TPoint

Visibility: public

Access: Read,Write

TDataModule.DesignSize

Declaration: Property DesignSize : TPoint

Visibility: public

Access: Read,Write

TDataModule.OnCreate

Declaration: Property OnCreate : TNotifyEvent

Visibility: published

Access: Read,Write

TDataModule.OnDestroy

Declaration: Property OnDestroy : TNotifyEvent

Visibility: published

Access: Read,Write

TDataModule.OldCreateOrder

Declaration: Property OldCreateOrder : Boolean

Visibility: published

Access: Read,Write

2.37 TFiler**Description**

Class responsible for streaming of components.

Method overview

Page	Method	Description
237	DefineBinaryProperty	
237	DefineProperty	
237	SetRoot	Sets the root component which is being streamed.

Property overview

Page	Property	Access	Description
238	Ancestor	rw	Ancestor component from which an inherited component is streamed.
238	IgnoreChildren	rw	Determines whether children will be streamed as well.
238	LookupRoot	r	Component used to look up ancestor components.
237	Root	rw	The root component is the initial component which is being streamed.

TFile.SetRoot

Synopsis: Sets the root component which is being streamed.

Declaration: `procedure SetRoot(ARoot: TComponent); Virtual`

Visibility: `protected`

Description: Sets the root component. The root component is the initial component which is being streamed.

TFile.DefineProperty

Synopsis:

Declaration: `procedure DefineProperty(const Name: String; ReadData: TReaderProc;
WriteData: TWriterProc; HasData: Boolean)
; Virtual; Abstract`

Visibility: `public`

Description:

TFile.DefineBinaryProperty

Synopsis:

Declaration: `procedure DefineBinaryProperty(const Name: String; ReadData: TStreamProc;
WriteData: TStreamProc; HasData: Boolean)
; Virtual; Abstract`

Visibility: `public`

Description:

TFile.Root

Synopsis: The root component is the initial component which is being streamed.

Declaration: `Property Root : TComponent`

Visibility: `public`

Access: `Read, Write`

Description: The streaming process will stream a component and all the components which it owns. The Root component is the component which is initially streamed.

See also: `TFile.LookupRoot` ([238](#))

TFile.LookupRoot

Synopsis: Component used to look up ancestor components.

Declaration: `Property LookupRoot : TComponent`

Visibility: `public`

Access: `Read`

Description: When comparing inherited component's values against parent values, the values are compared with the component in `LookupRoot`. Initially, it is set to `Root` (237).

See also: `TFile.Root` (237)

TFile.Ancestor

Synopsis: Ancestor component from which an inherited component is streamed.

Declaration: `Property Ancestor : TPersistent`

Visibility: `public`

Access: `Read,Write`

Description: When streaming a component, this is the parent component. Only properties that differ from the parent's property value will be streamed.

See also: `TFile.Root` (237), `TFile.LookupRoot` (238)

TFile.IgnoreChildren

Synopsis: Determines whether children will be streamed as well.

Declaration: `Property IgnoreChildren : Boolean`

Visibility: `public`

Access: `Read,Write`

Description: By default, all children (i.e. owned objects) will also be streamed when streaming a component. This property can be used to prevent owned objects from being streamed.

2.38 TFileStream**Description**

`TFileStream` is a `TStream` (271) descendent that stores or reads it's data from a named file in the filesystem of the operating system.

To this end, it overrides some of the abstract methods in `TStream` and implements them for the case of files on disk, and it adds the `FileName` (239) property to the list of public properties.

Method overview

Page	Method	Description
239	Create	Creates a file stream.
239	Destroy	Destroys the file stream.

Property overview

Page	Property	Access	Description
239	FileName	r	The filename of the stream.

TFileStream.Create

Synopsis: Creates a file stream.

Declaration: `constructor Create(const AFileName: String; Mode: Word)`
`constructor Create(const AFileName: String; Mode: Word; Rights: Cardinal)`

Visibility: public

Description: `Create` creates a new instance of a `TFileStream` class. It opens the file `AFileName` with mode `Mode`, which can have one of the following values:

Table 2.14:

<code>fmCreate</code>	<code>TFileStream.Create</code> (239) creates a new file if needed.
<code>fmOpenRead</code>	<code>TFileStream.Create</code> (239) opens a file with read-only access.
<code>fmOpenWrite</code>	<code>TFileStream.Create</code> (239) opens a file with write-only access.
<code>fmOpenReadWrite</code>	<code>TFileStream.Create</code> (239) opens a file with read-write access.

After the file has been opened in the requested mode and a handle has been obtained from the operating system, the inherited constructor is called.

Errors: If the file could not be opened in the requested mode, an `EFOpenError` ([172](#)) exception is raised.

See also: `TStream` ([271](#)), `TFileStream.FileName` ([239](#)), `THandleStream.Create` ([240](#))

TFileStream.Destroy

Synopsis: Destroys the file stream.

Declaration: `destructor Destroy; Override`

Visibility: public

Description: `Destroy` closes the file (causing possible buffered data to be written to disk) and then calls the inherited destructor.

Do not call `destroy` directly, instead call the `Free` method. `Destroy` does not check whether `Self` is nil, while `Free` does.

See also: `TFileStream.Create` ([239](#))

TFileStream.FileName

Synopsis: The filename of the stream.

Declaration: `Property FileName : String`

Visibility: public

Access: Read

Description: `FileName` is the name of the file that the stream reads from or writes to. It is the name as passed in the constructor of the stream; it cannot be changed. To write to another file, the stream must be freed and created again with the new filename.

See also: `TFileStream.Create` ([239](#))

2.39 THandleStream

Description

`THandleStream` is an abstract descendent of the `TStream` ([271](#)) class that provides methods for a stream to handle all reading and writing to and from a handle, provided by the underlying OS. To this end, it overrides the `Read` ([241](#)) and `Write` ([241](#)) methods of `TStream`.

Remark:

- `THandleStream` does not obtain a handle from the OS by itself, it just handles reading and writing to such a handle by wrapping the system calls for reading and writing; Descendent classes should obtain a handle from the OS by themselves and pass it on in the inherited constructor.
- Contrary to Delphi, no seek is implemented for `THandleStream`, since pipes and sockets do not support this. The seek is implemented in descendent methods that support it.

Method overview

Page	Method	Description
240	<code>Create</code>	Create a handlestream from an OS Handle.
241	<code>Read</code>	Overrides standard read method.
241	<code>Seek</code>	
240	<code>SetSize</code>	
241	<code>Write</code>	Overrides standard write method.

Property overview

Page	Property	Access	Description
241	<code>Handle</code>	r	The OS handle of the stream.

THandleStream.SetSize

Declaration: `procedure SetSize(NewSize: LongInt); Override`
`procedure SetSize(NewSize: Int64); Override`

Visibility: protected

THandleStream.Create

Synopsis: Create a handlestream from an OS Handle.

Declaration: constructor `Create(AHandle: Integer)`

Visibility: public

Description: Create creates a new instance of a THandleStream class. It stores AHandle in an internal variable and then calls the inherited constructor.

See also: TStream ([271](#))

THandleStream.Read

Synopsis: Overrides standard read method.

Declaration: `function Read(var Buffer;Count: LongInt) : LongInt; Override`

Visibility: public

Description: Read implements the abstract Read ([272](#)) method of TStream. It uses the Handle ([241](#)) property to read the Count bytes into Buffer

If no error occurs while reading, the number of bytes actually read will be returned.

Errors: If the operating system reports an error while reading from the handle, -1 is returned.

See also: TStream.Read ([272](#)), THandleStream.Write ([241](#)), THandleStream.Handle ([241](#))

THandleStream.Write

Synopsis: Overrides standard write method.

Declaration: `function Write(const Buffer;Count: LongInt) : LongInt; Override`

Visibility: public

Description: Write implements the abstract Write ([273](#)) method of TStream. It uses the Handle ([241](#)) property to write the Count bytes from Buffer.

If no error occurs while writing, the number of bytes actually written will be returned.

Errors: If the operating system reports an error while writing to handle, -1 is returned.

See also: TStream.Read ([272](#)), THandleStream.Write ([241](#)), THandleStream.Handle ([241](#))

THandleStream.Seek

Declaration: `function Seek(Offset: Int64;Origin: TSeekOrigin) : Int64; Override`

Visibility: public

THandleStream.Handle

Synopsis: The OS handle of the stream.

Declaration: `Property Handle : Integer`

Visibility: public

Access: Read

Description: Handle represents the Operating system handle to which reading and writing is done. The handle can be read only, i.e. it cannot be set after the THandlestream instance was created. It should be passed to the constructor THandleStream.Create ([240](#))

See also: THandleStream ([240](#)), THandleStream.Create ([240](#))

2.40 TList

Description

`TList` is a class that can be used to manage collections of pointers. It introduces methods and properties to store the pointers, search in the list of pointers, sort them. It manages its memory by itself, no intervention for that is needed.

To manage collections of strings, it is better to use a `TStrings` (287) descendent such as `TStringList` (280). To manage general objects, a `TCollection` (206) class exists, from which a descendent can be made to manage collections of various kinds.

Method overview

Page	Method	Description
244	Add	Adds a new pointer to the list.
246	Assign	
244	Clear	Clears the pointer list.
244	Delete	Removes a pointer from the list.
243	Destroy	Destroys the list and releases the memory used to store the list elements.
244	Error	Raises an <code>EListError</code> (172) exception.
244	Exchange	Exchanges two pointers in the list.
245	Expand	Increases the capacity of the list if needed.
245	Extract	
245	First	Returns the first non-nil pointer in the list.
242	Get	
243	Grow	
245	IndexOf	Returns the index of a given pointer.
246	Insert	Inserts a new pointer in the list at a given position.
246	Last	Returns the last non-nil pointer in the list.
246	Move	Moves a pointer from one position in the list to another.
243	Notify	
247	Pack	Removes <code>Nil</code> pointers from the list and frees unused memory.
243	Put	
243	RaiseIndexError	
246	Remove	Removes a value from the list.
243	SetCapacity	
243	SetCount	
247	Sort	Sorts the pointers in the list.

Property overview

Page	Property	Access	Description
247	Capacity	rw	Current capacity (i.e. number of pointers that can be stored) of the list.
248	Count	rw	Current number of pointers in the list.
248	Items	rw	Provides access to the pointers in the list.
248	List	r	Memory array where pointers are stored.

TList.Get

Declaration: `function Get(Index: Integer) : Pointer`

Visibility: protected

TList.Grow

Declaration: procedure Grow; Virtual

Visibility: protected

TList.Put

Declaration: procedure Put(Index: Integer; Item: Pointer)

Visibility: protected

TList.Notify

Declaration: procedure Notify(Ptr: Pointer; Action: TListNotification); Virtual

Visibility: protected

TList.SetCapacity

Declaration: procedure SetCapacity(NewCapacity: Integer)

Visibility: protected

TList.SetCount

Declaration: procedure SetCount(NewCount: Integer)

Visibility: protected

TList.RaiseIndexError

Declaration: procedure RaiseIndexError(Index: Integer)

Visibility: protected

TList.Destroy

Synopsis: Destroys the list and releases the memory used to store the list elements.

Declaration: destructor Destroy; Override

Visibility: public

Description: Destroy destroys the list and releases the memory used to store the list elements. The elements themselves are in no way touched, i.e. any memory they point to must be explicitly released before calling the destructor.

TList.Add

Synopsis: Adds a new pointer to the list.

Declaration: `function Add(Item: Pointer) : Integer`

Visibility: `public`

Description: `Add` adds a new pointer to the list after the last pointer (i.e. at position `Count`, thus increasing the item count with 1. If the list is at full capacity, the capacity of the list is expanded, using the `Grow` (243) method.

To insert a pointer at a certain position in the list, use the `Insert` (246) method instead.

See also: `TList.Delete` (244), `TList.Grow` (243), `TList.Insert` (246)

TList.Clear

Synopsis: Clears the pointer list.

Declaration: `procedure Clear; Dynamic`

Visibility: `public`

Description: `Clear` removes all pointers from the list, and sets the capacity to 0, thus freeing any memory allocated to maintain the list.

See also: `TList.Destroy` (243)

TList.Delete

Synopsis: Removes a pointer from the list.

Declaration: `procedure Delete(Index: Integer)`

Visibility: `public`

Description: `Delete` removes the pointer at position `Index` from the list, shifting all following pointers one position up (or to the left).

The memory the pointer is pointing to is *not* deallocated.

TList.Error

Synopsis: Raises an `EListError` (172) exception.

Declaration: `procedure Error(const Msg: String; Data: Integer); Virtual`

Visibility: `public`

Description: `Error` raises an `EListError` (172) exception, with a message formatted with `Msg` and `Data`.

TList.Exchange

Synopsis: Exchanges two pointers in the list.

Declaration: `procedure Exchange(Index1: Integer; Index2: Integer)`

Visibility: `public`

Description: `Exchange` exchanges the pointers at positions `Index1` and `Index2`. Both pointers must be within the current range of the list, or an `EListError` (172) exception will be raised.

TList.Expand

Synopsis: Increases the capacity of the list if needed.

Declaration: `function Expand : TList`

Visibility: `public`

Description: `Expand` increases the capacity of the list if the current element count matches the current list capacity.

The capacity is increased according to the following algorithm:

- 1.If the capacity is less than 3, the capacity is increased with 4.
- 2.If the capacity is larger than 3 and less than 8, the capacity is increased with 8.
- 3.If the capacity is larger than 8, the capacity is increased with 16.

The return value is `Self`.

See also: `TList.Capacity` ([247](#))

TList.Extract

Declaration: `function Extract(item: Pointer) : Pointer`

Visibility: `public`

TList.First

Synopsis: Returns the first non-nil pointer in the list.

Declaration: `function First : Pointer`

Visibility: `public`

Description: `First` returns the value of the first non-nil pointer in the list.

If there are no pointers in the list or all pointers equal `Nil`, then `Nil` is returned.

See also: `TList.Last` ([246](#))

TList.IndexOf

Synopsis: Returns the index of a given pointer.

Declaration: `function IndexOf(Item: Pointer) : Integer`

Visibility: `public`

Description: `IndexOf` searches for the pointer `Item` in the list of pointers, and returns the index of the pointer, if found.

If no pointer with the value `Item` was found, -1 is returned.

TList.Insert

Synopsis: Inserts a new pointer in the list at a given position.

Declaration: `procedure Insert(Index: Integer; Item: Pointer)`

Visibility: `public`

Description: `Insert` inserts pointer `Item` at position `Index` in the list. All pointers starting from `Index` are shifted to the right.

If `Index` is not a valid position, then a `EListError` (172) exception is raised.

See also: `TList.Add` (244), `TList.Delete` (244)

TList.Last

Synopsis: Returns the last non-nil pointer in the list.

Declaration: `function Last : Pointer`

Visibility: `public`

Description: `Last` returns the value of the last non-nil pointer in the list.

If there are no pointers in the list or all pointers equal `Nil`, then `Nil` is returned.

See also: `TList.First` (245)

TList.Move

Synopsis: Moves a pointer from one position in the list to another.

Declaration: `procedure Move(CurIndex: Integer; NewIndex: Integer)`

Visibility: `public`

Description: `Move` moves the pointer at position `CurIndex` to position `NewIndex`. This is done by storing the value at position `CurIndex`, deleting the pointer at position `CurIndex`, and reinserting the value at position `NewIndex`.

If `CurIndex` or `NewIndex` are not inside the valid range of indices, an `EListError` (172) exception is raised.

See also: `TList.Exchange` (244)

TList.Assign

Declaration: `procedure Assign(Obj: TList)`

Visibility: `public`

TList.Remove

Synopsis: Removes a value from the list.

Declaration: `function Remove(Item: Pointer) : Integer`

Visibility: `public`

Description: Remove searches `Item` in the list, and, if it finds it, deletes the item from the list. Only the first occurrence of `Item` is removed.

See also: `TList.Delete` ([244](#)), `TList.IndexOf` ([245](#)), `Tlist.Insert` ([246](#))

TList.Pack

Synopsis: Removes `Nil` pointers from the list and frees unused memory.

Declaration: `procedure Pack`

Visibility: `public`

Description: `Pack` removes all `nil` pointers from the list. The capacity of the list is then set to the number of pointers in the list. This method can be used to free unused memory if the list has grown to very large sizes and has a lot of unneeded `nil` pointers in it.

See also: `TList.Clear` ([244](#))

TList.Sort

Synopsis: Sorts the pointers in the list.

Declaration: `procedure Sort(Compare: TListSortCompare)`

Visibility: `public`

Description: `Sort`> sorts the pointers in the list. Two pointers are compared by passing them to the `Compare` function. The result of this function determines how the pointers will be sorted:

- If the result of this function is negative, the first pointer is assumed to be 'less' than the second and will be moved before the second in the list.
- If the function result is positive, the first pointer is assumed to be 'greater than' the second and will be moved after the second in the list.
- If the function result is zero, the pointers are assumed to be 'equal' and no moving will take place.

The sort is done using a quicksort algorithm.

TList.Capacity

Synopsis: Current capacity (i.e. number of pointers that can be stored) of the list.

Declaration: `Property Capacity : Integer`

Visibility: `public`

Access: `Read,Write`

Description: `Capacity` contains the number of pointers the list can store before it starts to grow.

If a new pointer is added to the list using `add` ([244](#)) or `insert` ([246](#)), and there is not enough memory to store the new pointer, then the list will try to allocate more memory to store the new pointer. Since this is a time consuming operation, it is important that this operation be performed as little as possible. If it is known how many pointers there will be before filling the list, it is a good idea to set the capacity first before filling. This ensures that the list doesn't need to grow, and will speed up filling the list.

See also: `TList.SetCapacity` ([243](#)), `TList.Count` ([248](#))

TList.Count

Synopsis: Current number of pointers in the list.

Declaration: `Property Count : Integer`

Visibility: `public`

Access: `Read,Write`

Description: `Count` is the current number of (possibly `Nil`) pointers in the list. Since the list is zero-based, the index of the largest pointer is `Count-1`.

TList.Items

Synopsis: Provides access to the pointers in the list.

Declaration: `Property Items[Index: Integer]: Pointer; default`

Visibility: `public`

Access: `Read,Write`

Description: `Items` is used to access the pointers in the list. It is the default property of the `TList` class, so it can be omitted.

The list is zero-based, so `Index` must be in the range 0 to `Count-1`.

TList.List

Synopsis: Memory array where pointers are stored.

Declaration: `Property List : PPointerList`

Visibility: `public`

Access: `Read`

Description: `List` points to the memory space where the pointers are stored. This can be used to quickly copy the list of pointers to another location.

2.41 TMemoryStream**Description**

`TMemoryStream` is a `TStream` (271) descendent that stores its data in memory. It descends directly from `TCustomMemoryStream` (231) and implements the necessary to allocate and de-allocate memory directly from the heap. It implements the `Write` (251) method which is missing in `TCustomMemoryStream`.

`TMemoryStream` also introduces methods to load the contents of another stream or a file into the memory stream.

It is not necessary to do any memory management manually, as the stream will allocate or de-allocate memory as needed. When the stream is freed, all allocated memory will be freed as well.

Method overview

Page	Method	Description
249	Clear	Zeroes the position, capacity and size of the stream.
249	Destroy	Frees any allocated memory and destroys the memory stream.
250	LoadFromFile	Loads the contents of a file into memory.
250	LoadFromStream	Loads the contents of a stream into memory.
249	Realloc	Sets the new capacity for the memory stream
250	SetSize	Sets the size for the memory stream.
251	Write	Writes data to the stream's memory.

Property overview

Page	Property	Access	Description
251	Capacity	rw	Current capacity of the stream.

TMemoryStream.Realloc

Synopsis: Sets the new capacity for the memory stream

Declaration: `function Realloc(var NewCapacity: LongInt) : Pointer; Virtual`

Visibility: `protected`

Description: `SetCapacity` sets the capacity of the memory stream, i.e. does the actual allocation or de-allocation of memory for the stream. It allocates at least `NewCapacity` bytes on the heap, moves the current contents of the stream to this location (as much as fits in) and returns the new memory location. Extra allocated memory is not initialized, i.e. may contain garbage.

Memory is allocated in blocks of 4 Kb; this can be changed by overriding the method.

See also: `TMemoryStream.Capacity` ([251](#))

TMemoryStream.Destroy

Synopsis: Frees any allocated memory and destroys the memory stream.

Declaration: `destructor Destroy; Override`

Visibility: `public`

Description: `Free` clears the memory stream, thus in effect freeing any memory allocated for it, and then frees the memory stream.

TMemoryStream.Clear

Synopsis: Zeroes the position, capacity and size of the stream.

Declaration: `procedure Clear`

Visibility: `public`

Description: `Clear` sets the position and size to 0, and sets the capacity of the stream to 0, thus freeing all memory allocated for the stream.

See also: `TStream.Size` ([280](#)), `TStream.Position` ([280](#)), `TCustomMemoryStream.Memory` ([234](#))

TMemoryStream.LoadFromStream

Synopsis: Loads the contents of a stream into memory.

Declaration: `procedure LoadFromStream(Stream: TStream)`

Visibility: `public`

Description: `LoadFromStream` loads the contents of `Stream` into the memorybuffer of the stream. Any previous contents of the memory stream are overwritten. Memory is allocated as needed.

Remark: The `LoadFromStream` uses the `Size` (280) property of `Stream` to determine how much memory must be allocated. Some streams do not allow the stream size to be determined, so care must be taken when using this method.

This method will work much faster than the use of the `TStream.CopyFrom` (274) method:

```
Seek(0,soFromBeginning);
CopyFrom(Stream,Stream.Size);
```

because the `CopyFrom` method copies the contents in blocks, while `LoadFromStream` reads the contents of the stream as one big block.

Errors: If an error occurs when reading from the stream, an `EStreamError` (173) may occur.

See also: `TStream.CopyFrom` (274), `TMemoryStream.LoadFromFile` (250)

TMemoryStream.LoadFromFile

Synopsis: Loads the contents of a file into memory.

Declaration: `procedure LoadFromFile(const FileName: String)`

Visibility: `public`

Description: `LoadFromFile` loads the contents of the file with name `FileName` into the memory stream. The current contents of the memory stream is replaced by the contents of the file. Memory is allocated as needed.

The `LoadFromFile` method simply creates a filestream and then calls the `TMemoryStream.LoadFromStream` (250) method.

See also: `TMemoryStream.LoadFromStream` (250)

TMemoryStream.SetSize

Synopsis: Sets the size for the memory stream.

Declaration: `procedure SetSize(NewSize: LongInt); Override`

Visibility: `public`

Description: `SetSize` sets the size of the memory stream to `NewSize`. This will set the capacity of the stream to `NewSize` and correct the current position in the stream when needed.

See also: `TStream.Position` (280), `TStream.Size` (280)

TMemoryStream.Write

Synopsis: Writes data to the stream's memory.

Declaration: `function Write(const Buffer; Count: LongInt) : LongInt; Override`

Visibility: `public`

Description: `Write` writes `Count` bytes from `Buffer` to the stream's memory, starting at the current position in the stream. If more memory is needed than currently allocated, more memory will be allocated. Any contents in the memory stream at the current position will be overwritten. The function returns the number of bytes actually written (which should under normal circumstances always equal `Count`).

This method overrides the abstract `TStream.Write` ([273](#)) method.

Errors: If no more memory could be allocated, then an exception will be raised.

See also: `TCustomMemoryStream.Read` ([232](#))

TMemoryStream.Capacity

Synopsis: Current capacity of the stream.

Declaration: `Property Capacity : LongInt`

Visibility: `protected`

Access: `Read, Write`

Description: `Capacity` is the current capacity of the stream, this is the current size of the memory allocated to the stream. This is not necessarily equal to the size of the stream, but will always be larger than or equal to the size of the stream. When writing to the stream, the `TMemoryStream.Write` ([251](#)) sets the capacity to the needed value.

If a lot of write operations will occur, performance may be improved by setting the capacity to a large value, so less reallocations of memory will occur while writing to the stream.

See also: `TMemoryStream.ReAlloc` ([249](#))

2.42 TOwnedCollection**Method overview**

Page	Method	Description
251	<code>Create</code>	
251	<code>GetOwner</code>	

TOwnedCollection.GetOwner

Declaration: `function GetOwner : TPersistent; Override`

Visibility: `protected`

TOwnedCollection.Create

Declaration: `constructor Create(AOwner: TPersistent; AItemClass: TCollectionItemClass)`

Visibility: `public`

2.43 TParser

Description

Class to parse the contents of a stream containing text data.

Method overview

Page	Method	Description
253	CheckToken	Checks whether the token is of the given type.
253	CheckTokenSymbol	Checks whether the token equals the given symbol
252	Create	Creates a new parser instance.
252	Destroy	Destroys the parser instance.
253	Error	Raises an EParserError (173) exception with the given message
253	ErrorFmt	Raises an EParserError (173) exception and formats the message.
253	ErrorStr	Raises an EParserError (173) exception with the given message
253	HexToBinary	Writes hexadecimal data to the stream.
254	NextToken	Reads the next token and returns its type.
254	SourcePos	Returns the current position in the stream.
254	TokenComponentIdent	Checks whether the current token is a component identifier.
254	TokenFloat	Returns the current token as a float.
254	TokenInt	Returns the current token as an integer.
254	TokenString	Returns the current token as a string.
255	TokenSymbolIs	Returns True if the current token is a symbol.

Property overview

Page	Property	Access	Description
255	SourceLine	r	Current source linenumber.
255	Token	r	Contents of the current token.

TParser.Create

Synopsis: Creates a new parser instance.

Declaration: constructor Create(Stream: TStream)

Visibility: public

Description: Creates a new parser instance.

TParser.Destroy

Synopsis: Destroys the parser instance.

Declaration: destructor Destroy; Override

Visibility: public

Description: Destroys the parser instance.

TParser.CheckToken

Synopsis: Checks whether the token if of the given type.

Declaration: `procedure CheckToken(T: Char)`

Visibility: public

Description: Checks whether the token if of the given type.

TParser.CheckTokenSymbol

Synopsis: Checks whether the token equals the given symbol

Declaration: `procedure CheckTokenSymbol(const S: String)`

Visibility: public

Description: Checks whether the token equals the given symbol

TParser.Error

Synopsis: Raises an `EParserError` (173) exception with the given message

Declaration: `procedure Error(const Ident: String)`

Visibility: public

Description: Raises an `EParserError` (173) exception with the given message

TParser.ErrorFmt

Synopsis: Raises an `EParserError` (173) exception and formats the message.

Declaration: `procedure ErrorFmt(const Ident: String;const Args: Array[] of const)`

Visibility: public

Description: Raises an `EParserError` (173) exception and formats the message.

TParser.ErrorStr

Synopsis: Raises an `EParserError` (173) exception with the given message

Declaration: `procedure ErrorStr(const Message: String)`

Visibility: public

Description: Raises an `EParserError` (173) exception with the given message

TParser.HexToBinary

Synopsis: Writes hexadecimal data to the stream.

Declaration: `procedure HexToBinary(Stream: TStream)`

Visibility: public

Description: Writes hexadecimal data to the stream.

TParser.NextToken

Synopsis: Reads the next token and returns its type.

Declaration: `function NextToken : Char`

Visibility: `public`

Description: Reads the next token and returns its type.

TParser.SourcePos

Synopsis: Returns the current position in the stream.

Declaration: `function SourcePos : LongInt`

Visibility: `public`

Description: Returns the current position in the stream.

TParser.TokenComponentIdent

Synopsis: Checks whether the current token is a component identifier.

Declaration: `function TokenComponentIdent : String`

Visibility: `public`

Description: Checks whether the current token is a component identifier.

TParser.TokenFloat

Synopsis: Returns the current token as a float.

Declaration: `function TokenFloat : Extended`

Visibility: `public`

Description: Returns the current token as a float.

TParser.TokenInt

Synopsis: Returns the current token as an integer.

Declaration: `function TokenInt : LongInt`

Visibility: `public`

Description: Returns the current token as an integer.

TParser.TokenString

Synopsis: Returns the current token as a string.

Declaration: `function TokenString : String`

Visibility: `public`

Description: Returns the current token as a string.

TParser.TokenSymbols

Synopsis: Returns True if the current token is a symbol.

Declaration: `function TokenSymbolIs(const S: String) : Boolean`

Visibility: public

Description: Returns True if the current token is a symbol.

TParser.SourceLine

Synopsis: Current source linenumber.

Declaration: `Property SourceLine : Integer`

Visibility: public

Access: Read

Description: Current source linenumber.

TParser.Token

Synopsis: Contents of the current token.

Declaration: `Property Token : Char`

Visibility: public

Access: Read

Description: Contents of the current token.

2.44 TPersistent**Description**

`TPersistent` is the basic class for the streaming system. Since it is compiled in the `{ $M+ }` state, the compiler generates RTTI (Run-Time Type Information) for it and all classes that descend from it. This information can be used to stream all properties of classes.

It also introduces functionality to assign the contents of 2 classes to each other.

Method overview

Page	Method	Description
257	<code>Assign</code>	Assign the contents of one class to another.
256	<code>AssignTo</code>	Generic assignment function.
256	<code>DefineProperties</code>	Declare non-published properties that need to be streamed.
257	<code>Destroy</code>	Destroys the <code>TPersistent</code> instance.
258	<code>GetNamePath</code>	Returns a string that can be used to identify the class instance.
257	<code>GetOwner</code>	Returns the owner of the component.

TPersistent.AssignTo

Synopsis: Generic assignment function.

Declaration: `procedure AssignTo(Dest: TPersistent); Virtual`

Visibility: `protected`

Description: `AssignTo` is the generic function to assign the class' contents to another class. This method must be overridden by descendent classes to actually assign the content of the source instance to the destination instance.

The `TPersistent` (255) implementation of `AssignTo` raises an `EConvertError` exception. This is done for the following reason: If the source class doesn't know how to assign itself to the destination class (using `AssignTo`), the destination class may know how get the data from the source class (using `Assign` (257)). If all descendent methods are implemented correctly, then if neither of the two classes knows how to assign their contents to each other, execution will end up at `TPersistent.Assign` (257), which will simply execute

```
Dest.AssignTo(Self);
```

If neither of the classes knows how to assign to/from each other, then execution will end up at the `TPersistent` implementation of `AssignTo`, and an exception will be raised.

See also: `TPersistent.Assign` (257)

TPersistent.DefineProperties

Synopsis: Declare non-published properties that need to be streamed.

Declaration: `procedure DefineProperties(Filer: TFiler); Virtual`

Visibility: `protected`

Description: `DefineProperties` must be overridden by descendent classes to indicate to the streaming system which non-published properties must also be streamed.

The streaming systems stores only published properties in the stream. Sometimes it is necessary to store additional data in the stream, data which is not published. This can be done by overriding the `DefineProperties` method. The `Filer` object is the class that is responsible for writing all properties to the stream.

To define new properties, two methods of the `TFiler` (236) class should be used:

1. `DefineProperty` (237), to define a property which can be represented as text.
2. `DefineProperty` (237), to define a property which contains binary data.

On order for the streaming to work correctly, a call to the inherited `DefineProperties` is also needed, so ancestor objects also get the possibility to read or write their private data to the stream. Failure to call the inherited method will result in component properties not being streamed correctly.

See also: `TFiler.DefineProperties` (236), `TFiler` (236)

TPersistent.GetOwner

Synopsis: Returns the owner of the component.

Declaration: `function GetOwner : TPersistent; Dynamic`

Visibility: `protected`

Description: `GetOwner` returns the owning component of the classes instance. The `TPersistent` implementation of `GetOwner` returns `Nil`. `TComponent` (217) overrides this method.

See also: `TComponent` (217)

TPersistent.Destroy

Synopsis: Destroys the `TPersistent` instance.

Declaration: `destructor Destroy; Override`

Visibility: `public`

Description: `Destroy` disposes of the persistent object. This method should never be called directly. Instead the `Free` method should be used.

TPersistent.Assign

Synopsis: Assign the contents of one class to another.

Declaration: `procedure Assign(Source: TPersistent); Virtual`

Visibility: `public`

Description: `Assign` copies the contents of `Source` to `Self`, if the classes of the destination and source classes are compatible.

The `TPersistent` implementation of `Assign` does nothing but calling the `AssignTo` (256) method of source. This means that if the destination class does not know how to assign the contents of the source class, the source class instance is asked to assign itself to the destination class. This means that it is necessary to implement only one of the two methods so that two classes can be assigned to one another.

Remark: In general, a statement of the form

```
Destination:=Source;
```

(where `Destination` and `Source` are classes) does not achieve the same as a statement of the form

```
Destination.Assign(Source);
```

After the former statement, both `Source` and `Destination` will point to the same object. The latter statement will copy the *contents* of the `Source` class to the `Destination` class.

See also: `TPersistent.AssignTo` (256)

TPersistent.GetNamePath

Synopsis: Returns a string that can be used to identify the class instance.

Declaration: `function GetNamePath : String; Virtual`

Visibility: `public`

Description: `GetNamePath` returns a string that can be used to identify the class instance. This can be used to display a name for this instance in a Object designer.

`GetNamePath` constructs a name by recursively prepending the `Classname` of the Owner instance to the `Classname` of this instance, separated by a dot.

See also: `TPersistent.GetOwner` ([257](#))

2.45 TReader

Description

The `TReader` class is a reader class that implements generic component streaming capabilities, independent of the format of the data in the stream. It uses a driver class `TAbstractObjectReader` ([174](#)) to do the actual reading of data. The interface of the `TReader` class should be identical to the interface in Delphi.

Method overview

Page	Method	Description
262	BeginReferences	Initializes the component referencing mechanism.
262	CheckValue	Raises an exception if the next value in the stream is not of type Value
266	CopyValue	Copy a value to a writer.
261	Create	Creates a new reader class
262	DefineBinaryProperty	Reads a user-defined binary property from the stream.
262	DefineProperty	Reads a user-defined property from the stream.
261	Destroy	Destroys a reader class.
262	EndOfList	Returns true if the stream contains an end-of-list marker.
263	EndReferences	Finalizes the component referencing mechanism.
260	Error	Calls an installed error handler and passes it Message
260	FindMethod	Return the address of a published method.
263	FixupReferences	Tries to resolve all unresolved component references.
263	NextValue	Returns the type of the next value.
261	PropertyError	Skips a property value and raises an exception.
263	ReadBoolean	Reads a boolean from the stream.
263	ReadChar	Reads a character from the stream.
263	ReadCollection	Reads a collection from the stream.
264	ReadComponent	Starts reading a component from the stream.
264	ReadComponents	Starts reading child components from the stream.
261	ReadData	Reads the components data after it has been created.
264	ReadDate	Reads a date from the stream
264	ReadFloat	Reads a float from the stream.
264	ReadIdent	Reads an identifier from the stream.
265	ReadInt64	Reads a 64-bit integer from the stream.
265	ReadInteger	Reads an integer from the stream
265	ReadListBegin	Checks for the beginning of a list.
265	ReadListEnd	Checks for the end of a list.
261	ReadProperty	Read and process a property name
261	ReadPropValue	Reads a property value for PropInfo.
265	ReadRootComponent	Starts reading a root component.
264	ReadSingle	Reads a single-type real from the stream.
265	ReadString	Reads a string from the stream.
266	ReadValue	Reads the next value type from the stream.

Property overview

Page	Property	Access	Description
266	CanHandleExceptions	r	Indicates whether the reader is handling exceptions at this stage.
266	Driver	r	The driver in use for streaming the data.
268	OnAncestorNotFound	rw	Handler called when the ancestor component cannot be found.
268	OnCreateComponent	rw	Handler called when a component needs to be created.
267	OnError	rw	Handler called when an error occurs.
269	OnFindComponentClass	rw	Handler called when a component class reference needs to be found.
267	OnFindMethod	rw	Handler to find or change a method address.
267	OnPropertyNotFound	rw	
269	OnReadStringProperty	rw	
268	OnReferenceName	rw	Handler called when another component is referenced.
268	OnSetMethodProperty	rw	
268	OnSetName	rw	Handler called when setting a component name.
267	Owner	rw	Owner of the component being read
267	Parent	rw	Parent of the component being read.
266	PropName	r	Name of the property being read at this moment.

TReader.Error

Synopsis: Calls an installed error handler and passes it Message

Declaration: `function Error(const Message: String) : Boolean; Virtual`

Visibility: protected

Description: Error returns False if no TReader.OnError ([267](#)) handler is installed. If one is installed, then it will be called, passing the reader instance, message, and function return value as parameters.

If the function result False, i.e. when there is no handler installed or the handler returned False, then the calling code will raise an exception.

See also: TReader.FindMethod ([260](#))

TReader.FindMethod

Synopsis: Return the address of a published method.

Declaration: `function FindMethod(ARoot: TComponent; const AMethodName: String)
: Pointer; Virtual`

Visibility: protected

Description: FindMethod will search for the method in ARoot. If it isn't found there, then it will call a OnFindMethod handler, if one is installed, passing it the method name AMethodName, the result pointer and a variable which says whether an exception should be raised if no method with name AMethodName is found.

If the method cannot be found and the OnFindMethod ([267](#)) returns True, then an exception will be raised.

See also: TReader.OnFindMethod ([267](#)), TFindMethodEvent ([155](#))

TReader.ReadProperty

Synopsis: Read and process a property name

Declaration: `procedure ReadProperty(AInstance: TPersistent)`

Visibility: `protected`

Description: Read and process a property name

TReader.ReadPropValue

Synopsis: Reads a property value for PropInfo.

Declaration: `procedure ReadPropValue(Instance: TPersistent; PropInfo: Pointer)`

Visibility: `protected`

Description: Reads a property value for PropInfo.

TReader.PropertyError

Synopsis: Skips a property value and raises an exception.

Declaration: `procedure PropertyError`

Visibility: `protected`

Description: Skips a property value and raises an exception.

TReader.ReadData

Synopsis: Reads the components data after it has been created.

Declaration: `procedure ReadData(Instance: TComponent)`

Visibility: `protected`

Description: Reads the components data after it has been created.

TReader.Create

Synopsis: Creates a new reader class

Declaration: `constructor Create(Stream: TStream; BufSize: Integer)`

Visibility: `public`

Description: Creates a new reader class

TReader.Destroy

Synopsis: Destroys a reader class.

Declaration: `destructor Destroy; Override`

Visibility: `public`

Description: Destroys a reader class.

TReader.BeginReferences

Synopsis: Initializes the component referencing mechanism.

Declaration: `procedure BeginReferences`

Visibility: `public`

Description: Initializes the component referencing mechanism.

TReader.CheckValue

Synopsis: Raises an exception if the next value in the stream is not of type `Value`

Declaration: `procedure CheckValue(Value: TValueType)`

Visibility: `public`

Description: Raises an exception if the next value in the stream is not of type `Value`

TReader.DefineProperty

Synopsis: Reads a user-defined property from the stream.

Declaration: `procedure DefineProperty(const Name: String; AReadData: TReaderProc;
WriteData: TWriterProc; HasData: Boolean)
; Override`

Visibility: `public`

Description: Reads a user-defined property from the stream.

TReader.DefineBinaryProperty

Synopsis: Reads a user-defined binary property from the stream.

Declaration: `procedure DefineBinaryProperty(const Name: String;
AReadData: TStreamProc;
WriteData: TStreamProc; HasData: Boolean)
; Override`

Visibility: `public`

Description: Reads a user-defined binary property from the stream.

TReader.EndOfList

Synopsis: Returns true if the stream contains an end-of-list marker.

Declaration: `function EndOfList : Boolean`

Visibility: `public`

Description: Returns true if the stream contains an end-of-list marker.

TReader.EndReferences

Synopsis: Finalizes the component referencing mechanism.

Declaration: `procedure EndReferences`

Visibility: `public`

Description: Finalizes the component referencing mechanism.

TReader.FixupReferences

Synopsis: Tries to resolve all unresolved component references.

Declaration: `procedure FixupReferences`

Visibility: `public`

Description: Tries to resolve all unresolved component references.

TReader.NextValue

Synopsis: Returns the type of the next value.

Declaration: `function NextValue : TValueType`

Visibility: `public`

Description: Returns the type of the next value.

TReader.ReadBoolean

Synopsis: Reads a boolean from the stream.

Declaration: `function ReadBoolean : Boolean`

Visibility: `public`

Description: Reads a boolean from the stream.

TReader.ReadChar

Synopsis: Reads a character from the stream.

Declaration: `function ReadChar : Char`

Visibility: `public`

Description: Reads a character from the stream.

TReader.ReadCollection

Synopsis: Reads a collection from the stream.

Declaration: `procedure ReadCollection(Collection: TCollection)`

Visibility: `public`

Description: Reads a collection from the stream.

TReader.ReadComponent

Synopsis: Starts reading a component from the stream.

Declaration: `function ReadComponent(Component: TComponent) : TComponent`

Visibility: public

Description: Starts reading a component from the stream.

TReader.ReadComponents

Synopsis: Starts reading child components from the stream.

Declaration: `procedure ReadComponents(AOwner: TComponent; AParent: TComponent;
Proc: TReadComponentsProc)`

Visibility: public

Description: Starts reading child components from the stream.

TReader.ReadFloat

Synopsis: Reads a float from the stream.

Declaration: `function ReadFloat : Extended`

Visibility: public

Description: Reads a float from the stream.

TReader.ReadSingle

Synopsis: Reads a single-type real from the stream.

Declaration: `function ReadSingle : Single`

Visibility: public

Description: Reads a single-type real from the stream.

TReader.ReadDate

Synopsis: Reads a date from the stream

Declaration: `function ReadDate : TDateTime`

Visibility: public

Description: Reads a date from the stream

TReader.ReadIdent

Synopsis: Reads an identifier from the stream.

Declaration: `function ReadIdent : String`

Visibility: public

Description: Reads an identifier from the stream.

TReader.ReadInteger

Synopsis: Reads an integer from the stream

Declaration: `function ReadInteger : LongInt`

Visibility: `public`

Description: Reads an integer from the stream

TReader.ReadInt64

Synopsis: Reads a 64-bit integer from the stream.

Declaration: `function ReadInt64 : Int64`

Visibility: `public`

Description: Reads a 64-bit integer from the stream.

TReader.ReadListBegin

Synopsis: Checks for the beginning of a list.

Declaration: `procedure ReadListBegin`

Visibility: `public`

Description: Checks for the beginning of a list.

TReader.ReadListEnd

Synopsis: Checks for the end of a list.

Declaration: `procedure ReadListEnd`

Visibility: `public`

Description: Checks for the end of a list.

TReader.ReadRootComponent

Synopsis: Starts reading a root component.

Declaration: `function ReadRootComponent (ARoot: TComponent) : TComponent`

Visibility: `public`

Description: Starts reading a root component.

TReader.ReadString

Synopsis: Reads a string from the stream.

Declaration: `function ReadString : String`

Visibility: `public`

Description: Reads a string from the stream.

TReader.ReadValue

Synopsis: Reads the next value type from the stream.

Declaration: `function ReadValue : TValueType`

Visibility: `public`

Description: Reads the next value type from the stream.

TReader.CopyValue

Synopsis: Copy a value to a writer.

Declaration: `procedure CopyValue(Writer: TWriter)`

Visibility: `public`

Description: Copy a value to a writer.

TReader.PropName

Synopsis: Name of the property being read at this moment.

Declaration: `Property PropName : String`

Visibility: `protected`

Access: `Read`

Description: Name of the property being read at this moment.

TReader.CanHandleExceptions

Synopsis: Indicates whether the reader is handling exceptions at this stage.

Declaration: `Property CanHandleExceptions : Boolean`

Visibility: `protected`

Access: `Read`

Description: Indicates whether the reader is handling exceptions at this stage.

TReader.Driver

Synopsis: The driver in use for streaming the data.

Declaration: `Property Driver : TAbstractObjectReader`

Visibility: `public`

Access: `Read`

Description: The driver in use for streaming the data.

TReader.Owner

Synopsis: Owner of the component being read

Declaration: Property Owner : TComponent

Visibility: public

Access: Read,Write

Description: Owner of the component being read

TReader.Parent

Synopsis: Parent of the component being read.

Declaration: Property Parent : TComponent

Visibility: public

Access: Read,Write

Description: Parent of the component being read.

TReader.OnError

Synopsis: Handler called when an error occurs.

Declaration: Property OnError : TReaderError

Visibility: public

Access: Read,Write

Description: Handler called when an error occurs.

TReader.OnPropertyNotFound

Declaration: Property OnPropertyNotFound : TPropertyNotFoundEvent

Visibility: public

Access: Read,Write

TReader.OnFindMethod

Synopsis: Handler to find or change a method address.

Declaration: Property OnFindMethod : TFindMethodEvent

Visibility: public

Access: Read,Write

Description: Handler to find or change a method address.

TReader.OnSetMethodProperty

Declaration: Property OnSetMethodProperty : TSetMethodPropertyEvent

Visibility: public

Access: Read,Write

TReader.OnSetName

Synopsis: Handler called when setting a component name.

Declaration: Property OnSetName : TSetNameEvent

Visibility: public

Access: Read,Write

Description: Handler called when setting a component name.

TReader.OnReferenceName

Synopsis: Handler called when another component is referenced.

Declaration: Property OnReferenceName : TReferenceNameEvent

Visibility: public

Access: Read,Write

Description: Handler called when another component is referenced.

TReader.OnAncestorNotFound

Synopsis: Handler called when the ancestor component cannot be found.

Declaration: Property OnAncestorNotFound : TAncestorNotFoundEvent

Visibility: public

Access: Read,Write

Description: Handler called when the ancestor component cannot be found.

TReader.OnCreateComponent

Synopsis: Handler called when a component needs to be created.

Declaration: Property OnCreateComponent : TCreateComponentEvent

Visibility: public

Access: Read,Write

Description: Handler called when a component needs to be created.

TReader.OnFindComponentClass

Synopsis: Handler called when a component class reference needs to be found.

Declaration: Property OnFindComponentClass : TFindComponentClassEvent

Visibility: public

Access: Read,Write

Description: Handler called when a component class reference needs to be found.

TReader.OnReadStringProperty

Declaration: Property OnReadStringProperty : TReadWriteStringPropertyEvent

Visibility: public

Access: Read,Write

2.46 TRecall

Method overview

Page	Method	Description
269	Create	
269	Destroy	
270	Forget	
269	Store	

Property overview

Page	Property	Access	Description
270	Reference	r	

TRecall.Create

Declaration: constructor Create(AStorage: TPersistent; AReference: TPersistent)

Visibility: public

TRecall.Destroy

Declaration: destructor Destroy; Override

Visibility: public

TRecall.Store

Declaration: procedure Store

Visibility: public

TRecall.Forget

Declaration: procedure Forget

Visibility: public

TRecall.Reference

Declaration: Property Reference : TPersistent

Visibility: public

Access: Read

2.47 TResourceStream**Description**

Stream that reads its data from a resource object.

Method overview

Page	Method	Description
270	Create	Creates a new instance of a resource stream.
270	CreateFromID	Creates a new instance of a resource stream with resource
271	Destroy	Destroys the instance of the resource stream.
271	Write	Write implements the abstract TStream.Write (273) method.

TResourceStream.Create

Synopsis: Creates a new instance of a resource stream.

Declaration: constructor Create(Instance: THandle; const ResName: String;
ResType: PChar)

Visibility: public

Description: Creates a new instance of a resource stream.

TResourceStream.CreateFromID

Synopsis: Creates a new instance of a resource stream with resource

Declaration: constructor CreateFromID(Instance: THandle; ResID: Integer;
ResType: PChar)

Visibility: public

Description: Creates a new instance of a resource stream with resource

TResourceStream.Destroy

Synopsis: Destroys the instance of the resource stream.

Declaration: destructor Destroy; Override

Visibility: public

Description: Destroys the instance of the resource stream.

TResourceStream.Write

Synopsis: Write implements the abstract TStream.Write ([273](#)) method.

Declaration: function Write(const Buffer;Count: LongInt) : LongInt; Override

Visibility: public

Description: Write implements the abstract TStream.Write ([273](#)) method.

2.48 TStream**Description**

TStream is the base class for all streaming classes. It defines abstract methods for reading ([272](#)), writing ([273](#)) from and to streams, as well as functions to determine the size of the stream as well as the current position of the stream.

Descendent classes such as TMemoryStream ([248](#)) or TFileStream ([238](#)) then implement these abstract methods to write streams to memory or file.

Method overview

Page	Method	Description
274	CopyFrom	Copy data from one stream to another
277	FixupResourceHeader	Not implemented in FPC
272	Read	Reads data from the stream to a buffer and returns the number of bytes read.
278	ReadAnsiString	Read an ansistring from the stream and return its value.
274	ReadBuffer	Reads data from the stream to a buffer
277	ReadByte	Read a byte from the stream and return its value.
275	ReadComponent	Reads component data from a stream
275	ReadComponentRes	Reads component data and resource header from a stream
278	ReadDWord	Read a DWord from the stream and return its value.
277	ReadResHeader	Read a resource header from the stream.
277	ReadWord	Read a word from the stream and return its value.
273	Seek	Sets the current position in the stream
272	SetSize	Sets the size of the stream
273	Write	Writes data from the stream to the buffer and returns the number of bytes written.
279	WriteAnsiString	Write an ansistring to the stream.
274	WriteBuffer	Writes data from the stream to the buffer
278	WriteByte	Write a byte to the stream.
275	WriteComponent	Write component data to the stream
276	WriteComponentRes	Write resource header and component data to a stream
276	WriteDescendent	Write component data to a stream, relative to an ancestor
276	WriteDescendentRes	Write resource header and component data to a stream, relative to an ancestor
279	WriteDWord	Write a DWord to the stream.
276	WriteResourceHeader	Write resource header to the stream
279	WriteWord	Write a word to the stream.

Property overview

Page	Property	Access	Description
280	Position	rw	The current position in the stream.
280	Size	rw	The current size of the stream.

TStream.SetSize

Synopsis: Sets the size of the stream

Declaration: `procedure SetSize(NewSize: LongInt); Virtual; Overload`
`procedure SetSize(NewSize: Int64); Virtual; Overload`

Visibility: protected

Description: `SetSize` is the write handler for the `TStream.Size` ([280](#)) property. The `TStream` implementation of `SetSize` does nothing, but descendent classes may override this methods to allow programmers to set the size of the stream.

See also: `TStream.GetSize` ([271](#)), `TStream.Size` ([280](#))

TStream.Read

Synopsis: Reads data from the stream to a buffer and returns the number of bytes read.

Declaration: `function Read(var Buffer;Count: LongInt) : LongInt; Virtual; Abstract`

Visibility: public

Description: Read attempts to read Count from the stream to Buffer and returns the number of bytes actually read.

This method should be used when the number of bytes is not determined. If a specific number of bytes is expected, use TStream.ReadBuffer (274) instead.

Read is an abstract method that is overridden by descendent classes to do the actual reading.

Errors: Descendent classes that do not allow reading from the stream may raise an exception when the Read is used.

See also: TStream.Write (273), TStream.ReadBuffer (274)

TStream.Write

Synopsis: Writes data from the stream to the buffer and returns the number of bytes written.

Declaration: `function Write(const Buffer;Count: LongInt) : LongInt; Virtual
; Abstract`

Visibility: public

Description: Write attempts to write Count bytes from Buffer to the stream. It returns the actual number of bytes written to the stream.

This method should be used when the number of bytes that should be written is not determined. If a specific number of bytes should be written, use TStream.WriteBuffer (274) instead.

Write is an abstract method that is overridden by descendent classes to do the actual writing.

Errors: Descendent classes that do not allow writing to the stream may raise an exception when Write is used.

See also: TStream.Read (272), TStream.WriteBuffer (274)

TStream.Seek

Synopsis: Sets the current position in the stream

Declaration: `function Seek(Offset: LongInt;Origin: Word) : LongInt; Virtual
; Overload
function Seek(Offset: Int64;Origin: TSeekOrigin) : Int64; Virtual
; Overload`

Visibility: public

Description: Seek sets the position of the stream to Offset bytes from Origin. Origin can have one of the following values:

Offset should be negative when the origin is SoFromEnd. It should be positive for soFromBeginning and can have both signs for soFromCurrent

This is an abstract method, which must be overridden by descendent classes. They may choose not to implement this method for all values of Origin and Offset.

Errors: An exception may be raised if this method is called with an invalid pair of Offset,Origin values. e.g. a negative offset for soFromBeginning.

See also: TStream.Position (280)

Table 2.15:

Constant	Meaning
<code>soFromBeginning</code>	Set the position relative to the start of the stream.
<code>soFromCurrent</code>	Set the position relative to the beginning of the stream.
<code>soFromEnd</code>	Set the position relative to the end of the stream.

TStream.ReadBuffer

Synopsis: Reads data from the stream to a buffer

Declaration: `procedure ReadBuffer(var Buffer; Count: LongInt)`

Visibility: `public`

Description: `ReadBuffer` reads `Count` bytes of the stream into `Buffer`. If the stream does not contain `Count` bytes, then an exception is raised.

`ReadBuffer` should be used to read in a fixed number of bytes, such as when reading structures or the content of variables. If the number of bytes is not determined, use `TStream.Read` (272) instead. `ReadBuffer` uses `Read` internally to do the actual reading.

Errors: If the stream does not allow to read `Count` bytes, then an exception is raised.

See also: `TStream.Read` (272), `TStream.WriteBuffer` (274)

TStream.WriteBuffer

Synopsis: Writes data from the stream to the buffer

Declaration: `procedure WriteBuffer(const Buffer; Count: LongInt)`

Visibility: `public`

Description: `WriteBuffer` writes `Count` bytes to the stream from `Buffer`. If the stream does not allow `Count` bytes to be written, then an exception is raised.

`WriteBuffer` should be used to read in a fixed number of bytes, such as when writing structures or the content of variables. If the number of bytes is not determined, use `TStream.Write` (273) instead. `WriteBuffer` uses `Write` internally to do the actual reading.

Errors: If the stream does not allow to write `Count` bytes, then an exception is raised.

See also: `TStream.Write` (273), `TStream.ReadBuffer` (274)

TStream.CopyFrom

Synopsis: Copy data from one stream to another

Declaration: `function CopyFrom(Source: TStream; Count: Int64) : Int64`

Visibility: `public`

Description: `CopyFrom` reads `Count` bytes from `Source` and writes them to the current stream. This updates the current position in the stream. After the action is completed, the number of bytes copied is returned.

This can be used to quickly copy data from one stream to another or to copy the whole contents of the stream.

See also: `TStream.Read` (272), `TStream.Write` (273)

TStream.ReadComponent

Synopsis: Reads component data from a stream

Declaration: `function ReadComponent(Instance: TComponent) : TComponent`

Visibility: public

Description: `ReadComponent` reads a component state from the stream and transfers this state to `Instance`. If `Instance` is `nil`, then it is created first based on the type stored in the stream. `ReadComponent` returns the component as it is read from the stream.

`ReadComponent` simply creates a `TReader` (258) object and calls its `ReadRootComponent` (265) method.

Errors: If an error occurs during the reading of the component, an `EFileError` (172) exception is raised.

See also: `TStream.WriteComponent` (275), `TStream.ReadComponentRes` (275), `TReader.ReadRootComponent` (265)

TStream.ReadComponentRes

Synopsis: Reads component data and resource header from a stream

Declaration: `function ReadComponentRes(Instance: TComponent) : TComponent`

Visibility: public

Description: `ReadComponentRes` reads a resource header from the stream, and then calls `ReadComponent` (275) to read the component state from the stream into `Instance`.

This method is usually called by the global streaming method when instantiating forms and datamodules as created by an IDE. It should be used mainly on Windows, to store components in Windows resources.

Errors: If an error occurs during the reading of the component, an `EFileError` (172) exception is raised.

See also: `TStream.ReadComponent` (275), `TStream.WriteComponentRes` (276)

TStream.WriteComponent

Synopsis: Write component data to the stream

Declaration: `procedure WriteComponent(Instance: TComponent)`

Visibility: public

Description: `WriteComponent` writes the published properties of `Instance` to the stream, so they can later be read with `TStream.ReadComponent` (275). This method is intended to be used by an IDE, to preserve the state of a form or datamodule as designed in the IDE.

`WriteComponent` simply calls `WriteDescendent` (276) with `Nil` ancestor.

See also: `TStream.ReadComponent` (275), `TStream.WriteComponentRes` (276)

TStream.WriteComponentRes

Synopsis: Write resource header and component data to a stream

Declaration: `procedure WriteComponentRes(const ResName: String; Instance: TComponent)`

Visibility: public

Description: `WriteComponentRes` writes a `ResName` resource header to the stream and then calls `WriteComponent` (275) to write the published properties of `Instance` to the stream.

This method is intended for use by an IDE that can use it to store forms or datamodules as designed in a Windows resource stream.

See also: `TStream.WriteComponent` (275), `TStream.ReadComponentRes` (275)

TStream.WriteDescendent

Synopsis: Write component data to a stream, relative to an ancestor

Declaration: `procedure WriteDescendent(Instance: TComponent; Ancestor: TComponent)`

Visibility: public

Description: `WriteDescendent` writes the state of `Instance` to the stream where it differs from `Ancestor`, i.e. only the changed properties are written to the stream.

`WriteDescendent` creates a `TWriter` (311) object and calls its `WriteDescendent` (314) object. The writer is passed a binary driver object (196) by default.

TStream.WriteDescendentRes

Synopsis: Write resource header and component data to a stream, relative to an ancestor

Declaration: `procedure WriteDescendentRes(const ResName: String; Instance: TComponent; Ancestor: TComponent)`

Visibility: public

Description: `WriteDescendentRes` writes a `ResName` resource header, and then calls `WriteDescendent` (276) to write the state of `Instance` to the stream where it differs from `Ancestor`, i.e. only the changed properties are written to the stream.

This method is intended for use by an IDE that can use it to store forms or datamodules as designed in a Windows resource stream.

TStream.WriteResourceHeader

Synopsis: Write resource header to the stream

Declaration: `procedure WriteResourceHeader(const ResName: String; var FixupInfo: Integer)`

Visibility: public

Description: `WriteResourceHeader` writes a resource-file header for a resource called `ResName`. It returns in `FixupInfo` the argument that should be passed on to `TStream.FixupResourceHeader` (277).

`WriteResourceHeader` should not be used directly. It is called by the `TStream.WriteComponentRes` (276) and `TStream.WriteDescendentRes` (276) methods.

See also: `TStream.FixupResourceHeader` (277), `TStream.WriteComponentRes` (276), `TStream.WriteDescendentRes` (276)

TStream.FixupResourceHeader

Synopsis: Not implemented in FPC

Declaration: `procedure FixupResourceHeader(FixupInfo: Integer)`

Visibility: `public`

Description: `FixupResourceHeader` is used to write the size of the resource after a component was written to stream. The size is determined from the current position, and it is written at position `FixupInfo`. After that the current position is restored.

`FixupResourceHeader` should never be called directly; it is handled by the streaming system.

See also: `TStream.WriteResourceHeader` (276), `TStream.WriteComponentRes` (276), `TStream.WriteDescendentRes` (276)

TStream.ReadResHeader

Synopsis: Read a resource header from the stream.

Declaration: `procedure ReadResHeader`

Visibility: `public`

Description: `ReadResourceHeader` reads a resource file header from the stream. It positions the stream just beyond the header.

`ReadResourceHeader` should not be called directly, it is called by the streaming system when needed.

Errors: If the resource header is invalid an `EInvalidImage` (172) exception is raised.

See also: `TStream.ReadComponentRes` (275), `EInvalidImage` (172)

TStream.ReadByte

Synopsis: Read a byte from the stream and return its value.

Declaration: `function ReadByte : Byte`

Visibility: `public`

Description: `ReadByte` reads one byte from the stream and returns its value.

Errors: If the byte cannot be read, a `EStreamError` (173) exception will be raised. This is a utility function which simply calls the `Read` (272) function.

See also: `TStream.Read` (272), `TStream.WriteByte` (278), `TStream.ReadWord` (277), `TStream.ReadDWord` (278), `TStream.ReadAnsiString` (278)

TStream.ReadWord

Synopsis: Read a word from the stream and return its value.

Declaration: `function ReadWord : Word`

Visibility: `public`

Description: ReadWord reads one Word (i.e. 2 bytes) from the stream and returns its value. This is a utility function which simply calls the Read (272) function.

Errors: If the word cannot be read, a EStreamError (173) exception will be raised.

See also: TStream.Read (272), TStream.WriteWord (279), TStream.ReadByte (277), TStream.ReadDWord (278), TStream.ReadAnsiString (278)

TStream.ReadDWord

Synopsis: Read a DWord from the stream and return its value.

Declaration: function ReadDWord : Cardinal

Visibility: public

Description: ReadDWord reads one DWord (i.e. 4 bytes) from the stream and returns its value. This is a utility function which simply calls the Read (272) function.

Errors: If the DWord cannot be read, a EStreamError (173) exception will be raised.

See also: TStream.Read (272), TStream.WriteDWord (279), TStream.ReadByte (277), TStream.ReadWord (277), TStream.ReadAnsiString (278)

TStream.ReadAnsiString

Synopsis: Read an ansistring from the stream and return its value.

Declaration: function ReadAnsiString : String

Visibility: public

Description: ReadAnsiString reads an ansistring from the stream and returns its value. This is a utility function which simply calls the read function several times. The Ansistring should be stored as 4 bytes (a DWord) representing the length of the string, and then the string value itself. The WriteAnsiString (279) function writes an ansistring in such a format.

Errors: If the AnsiString cannot be read, a EStreamError (173) exception will be raised.

See also: TStream.Read (272), TStream.WriteAnsiString (279), TStream.ReadByte (277), TStream.ReadWord (277), TStream.ReadDWord (278)

TStream.WriteByte

Synopsis: Write a byte to the stream.

Declaration: procedure WriteByte(b: Byte)

Visibility: public

Description: WriteByte writes the byte B to the stream. This is a utility function which simply calls the Write (273) function. The byte can be read from the stream using the ReadByte (277) function.

Errors: If an error occurs when attempting to write, an EStreamError (173) exception will be raised.

See also: TStream.Write (273), TStream.ReadByte (277), TStream.WriteWord (279), TStream.WriteDWord (279), TStream.WriteAnsiString (279)

TStream.WriteWord

Synopsis: Write a word to the stream.

Declaration: `procedure WriteWord(w: Word)`

Visibility: `public`

Description: `WriteWord` writes the word `W` (i.e. 2 bytes) to the stream. This is a utility function which simply calls the `Write` (273) function. The word can be read from the stream using the `ReadWord` (277) function.

Errors: If an error occurs when attempting to write, an `EStreamError` (173) exception will be raised.

See also: `TStream.Write` (273), `TStream.ReadWord` (277), `TStream.WriteByte` (278), `TStream.WriteDWord` (279), `TStream.WriteAnsiString` (279)

TStream.WriteDWord

Synopsis: Write a DWord to the stream.

Declaration: `procedure WriteDWord(d: Cardinal)`

Visibility: `public`

Description: `WriteDWord` writes the DWord `D` (i.e. 4 bytes) to the stream. This is a utility function which simply calls the `Write` (273) function. The DWord can be read from the stream using the `ReadDWord` (278) function.

Errors: If an error occurs when attempting to write, an `EStreamError` (173) exception will be raised.

See also: `TStream.Write` (273), `TStream.ReadDWord` (278), `TStream.WriteByte` (278), `TStream.WriteWord` (279), `TStream.WriteAnsiString` (279)

TStream.WriteAnsiString

Synopsis: Write an ansistring to the stream.

Declaration: `procedure WriteAnsiString(S: String)`

Visibility: `public`

Description: `WriteAnsiString` writes the `AnsiString` `S` (i.e. 4 bytes) to the stream. This is a utility function which simply calls the `Write` (273) function. The ansistring is written as a 4 byte length specifier, followed by the ansistring's content. The ansistring can be read from the stream using the `ReadAnsiString` (278) function.

Errors: If an error occurs when attempting to write, an `EStreamError` (173) exception will be raised.

See also: `TStream.Write` (273), `TStream.ReadAnsiString` (278), `TStream.WriteByte` (278), `TStream.WriteWord` (279), `TStream.WriteDWord` (279)

TStream.Position

Synopsis: The current position in the stream.

Declaration: `Property Position : Int64`

Visibility: `public`

Access: `Read,Write`

Description: `Position` can be read to determine the current position in the stream. It can be written to to set the (absolute) position in the stream. The position is zero-based, so to set the position at the beginning of the stream, the position must be set to zero.

Remark: Not all `TStream` descendants support setting the position in the stream, so this should be used with care.

Errors: Some descendents may raise an `EStreamError` (173) exception if they do not support setting the stream position.

See also: `TStream.Size` (280), `TStream.Seek` (273)

TStream.Size

Synopsis: The current size of the stream.

Declaration: `Property Size : Int64`

Visibility: `public`

Access: `Read,Write`

Description: `Size` can be read to determine the stream size or to set the stream size.

Remark: Not all descendents of `TStream` support getting or setting the stream size; they may raise an exception if the `Size` property is read or set.

See also: `TStream.Position` (280), `TStream.Seek` (273)

2.49 TStringList**Description**

`TStringList` is a descendent class of `TStrings` (287) that implements all of the abstract methods introduced there. It also introduces some additional methods:

- Sort the list, or keep the list sorted at all times
- Special handling of duplicates in sorted lists
- Notification of changes in the list

Method overview

Page	Method	Description
283	Add	Implements the TStrings.Add (292) function.
281	Changed	Called when the list of strings was modified.
281	Changing	Called when the list is changing.
284	Clear	Implements the TStrings.Add (292) function.
285	CustomSort	
284	Delete	Implements the TStrings.Delete (294) function.
283	Destroy	Destroys the stringlist.
284	Exchange	Implements the TStrings.Exchange (295) function.
284	Find	Locates the index for a given string in sorted lists.
282	Get	Overrides the standard read handler for the TStrings.Strings (302) property.
282	GetCapacity	Overrides the standard read handler for the TStrings.Capacity (299) property.
282	GetCount	Overrides the standard read handler for the TStrings.Count (300) property.
282	GetObject	Overrides the standard read handler for the TStrings.Objects (301) property.
285	IndexOf	Overrides the TStrings.IndexOf (295) property.
285	Insert	Overrides the TStrings.Insert (296) method.
282	Put	Overrides the standard write handler for the TStrings.Strings (302) property.
283	PutObject	Overrides the standard write handler for the TStrings.Objects (301) property.
283	SetCapacity	Overrides the standard write handler for the TStrings.Capacity (299) property.
283	SetUpdateState	Overrides the standard TStrings.SetUpdateState (292) call.
285	Sort	Sorts the strings in the list.

Property overview

Page	Property	Access	Description
285	Duplicates	rw	Describes the behaviour of a sorted list with respect to duplicate strings.
286	OnChange	rw	Event triggered after the list was modified.
286	OnChanging	rw	Event triggered when the list is about to be modified.
286	Sorted	rw	Determines whether the list is sorted or not.

TStringList.Changed

Synopsis: Called when the list of strings was modified.

Declaration: `procedure Changed; Virtual`

Visibility: `protected`

Description: Called when the list of strings was modified.

TStringList.Changing

Synopsis: Called when the list is changing.

Declaration: `procedure Changing; Virtual`

Visibility: `protected`

Description: Called when the list is changing.

TStringList.Get

Synopsis: Overrides the standard read handler for the `TStrings.Strings` (302) property.

Declaration: `function Get(Index: Integer) : String; Override`

Visibility: `protected`

Description: Overrides the standard read handler for the `TStrings.Strings` (302) property.

TStringList.GetCapacity

Synopsis: Overrides the standard read handler for the `TStrings.Capacity` (299) property.

Declaration: `function GetCapacity : Integer; Override`

Visibility: `protected`

Description: Overrides the standard read handler for the `TStrings.Capacity` (299) property.

TStringList.GetCount

Synopsis: Overrides the standard read handler for the `TStrings.Count` (300) property.

Declaration: `function GetCount : Integer; Override`

Visibility: `protected`

Description: Overrides the standard read handler for the `TStrings.Count` (300) property.

TStringList.GetObject

Synopsis: Overrides the standard read handler for the `TStrings.Objects` (301) property.

Declaration: `function GetObject(Index: Integer) : TObject; Override`

Visibility: `protected`

Description: Overrides the standard read handler for the `TStrings.Objects` (301) property.

TStringList.Put

Synopsis: Overrides the standard write handler for the `TStrings.Strings` (302) property.

Declaration: `procedure Put(Index: Integer; const S: String); Override`

Visibility: `protected`

Description: Overrides the standard write handler for the `TStrings.Strings` (302) property.

TStringList.PutObject

Synopsis: Overrides the standard write handler for the TStrings.Objects (301) property.

Declaration: `procedure PutObject(Index: Integer; AObject: TObject); Override`

Visibility: `protected`

Description: Overrides the standard write handler for the TStrings.Objects (301) property.

TStringList.SetCapacity

Synopsis: Overrides the standard write handler for the TStrings.Capacity (299) property.

Declaration: `procedure SetCapacity(NewCapacity: Integer); Override`

Visibility: `protected`

Description: Overrides the standard write handler for the TStrings.Capacity (299) property.

TStringList.SetUpdateState

Synopsis: Overrides the standard TStrings.SetUpdateState (292) call.

Declaration: `procedure SetUpdateState(Updating: Boolean); Override`

Visibility: `protected`

Description: Overrides the standard TStrings.SetUpdateState (292) call.

TStringList.Destroy

Synopsis: Destroys the stringlist.

Declaration: `destructor Destroy; Override`

Visibility: `public`

Description: `Destroy` clears the stringlist, release all memory allocated for the storage of the strings, and then calls the inherited destroy method.

Remark: Any objects associated to strings in the list will *not* be destroyed; it is the responsibility of the caller to destroy all objects associated with strings in the list.

TStringList.Add

Synopsis: Implements the TStrings.Add (292) function.

Declaration: `function Add(const S: String) : Integer; Override`

Visibility: `public`

Description: `Add` will add `S` to the list. If the list is sorted and the string `S` is already present in the list and `TStringList.Duplicates` (285) is `dupError` then an `EStringListError` (174) exception is raised. If `Duplicates` is set to `dupIgnore` then the return value is undefined.

If the list is sorted, new strings will not necessarily be added to the end of the list, rather they will be inserted at their alphabetical position.

Errors: If the list is sorted and the string *S* is already present in the list and `TStringList.Duplicates` (285) is `dupError` then an `EStringListError` (174) exception is raised.

See also: `TStringList.Insert` (285), `TStringList.Duplicates` (285)

TStringList.Clear

Synopsis: Implements the `TStrings.Add` (292) function.

Declaration: `procedure Clear; Override`

Visibility: `public`

Description: Implements the `TStrings.Add` (292) function.

TStringList.Delete

Synopsis: Implements the `TStrings.Delete` (294) function.

Declaration: `procedure Delete(Index: Integer); Override`

Visibility: `public`

Description: Implements the `TStrings.Delete` (294) function.

TStringList.Exchange

Synopsis: Implements the `TStrings.Exchange` (295) function.

Declaration: `procedure Exchange(Index1: Integer; Index2: Integer); Override`

Visibility: `public`

Description: `Exchange` will exchange two items in the list as described in `TStrings.Exchange` (295).

Remark: `Exchange` will not check whether the list is sorted or not; if `Exchange` is called on a sorted list and the strings are not identical, the sort order of the list will be destroyed.

See also: `TStringList.Sorted` (286), `TStrings.Exchange` (295)

TStringList.Find

Synopsis: Locates the index for a given string in sorted lists.

Declaration: `function Find(const S: String; var Index: Integer) : Boolean; Virtual`

Visibility: `public`

Description: `Find` returns `True` if the string *S* is present in the list. Upon exit, the `Index` parameter will contain the position of the string in the list. If the string is not found, the function will return `False` and `Index` will contain the position where the string will be inserted if it is added to the list.

Remark:

1. Use this method only on sorted lists. For unsorted lists, use `TStringList.IndexOf` (285) instead.
2. `Find` uses a binary search method to locate the string

TStringList.IndexOf

Synopsis: Overrides the TStrings.IndexOf ([295](#)) property.

Declaration: `function IndexOf(const S: String) : Integer; Override`

Visibility: public

Description: IndexOf overrides the ancestor method TStrings.indexOf ([295](#)). It tries to optimize the search by executing a binary search if the list is sorted. The function returns the position of S if it is found in the list, or -1 if the string is not found in the list.

See also: TStrings.IndexOf ([295](#)), TStringList.Find ([284](#))

TStringList.Insert

Synopsis: Overrides the TStrings.Insert ([296](#)) method.

Declaration: `procedure Insert(Index: Integer;const S: String); Override`

Visibility: public

Description: Insert will insert the string S at position Index in the list. If the list is sorted, an EStringListError ([174](#)) exception will be raised instead. Index is a zero-based position.

Errors: If Index contains an invalid value (less than zero or larger than Count, or the list is sorted, an EStringListError ([174](#)) exception will be raised.

See also: TStringList.Add ([283](#)), TStrings.Insert ([296](#)), TStringList.InsertObject ([280](#))

TStringList.Sort

Synopsis: Sorts the strings in the list.

Declaration: `procedure Sort; Virtual`

Visibility: public

Description: Sort will sort the strings in the list using the quicksort algorithm. If the list has its TStringList.Sorted ([286](#)) property set to True then nothing will be done.

See also: TStringList.Sorted ([286](#))

TStringList.CustomSort

Declaration: `procedure CustomSort(CompareFn: TStringListSortCompare)`

Visibility: public

TStringList.Duplicates

Synopsis: Describes the behaviour of a sorted list with respect to duplicate strings.

Declaration: `Property Duplicates : TDuplicates`

Visibility: public

Access: Read,Write

Description: Duplicates describes what to do in case a duplicate value is added to the list:

If the stringlist is not sorted, the Duplicates setting is ignored.

Table 2.16:

dupIgnore	Duplicate values will not be added to the list, but no error will be triggered.
dupError	If an attempt is made to add a duplicate value to the list, an <code>EStringListError</code> (174) exception is raised.
dupAccept	Duplicate values can be added to the list.

TStringList.Sorted

Synopsis: Determines whether the list is sorted or not.

Declaration: Property Sorted : Boolean

Visibility: public

Access: Read,Write

Description: Sorted can be set to True in order to cause the list of strings to be sorted. Further additions to the list will be inserted at the correct position so the list remains sorted at all times. Setting the property to False has no immediate effect, but will allow strings to be inserted at any position.

Remark:

1. When Sorted is True, `TStringList.Insert` (285) cannot be used. For sorted lists, `TStringList.Add` (283) should be used instead.
2. If Sorted is True, the `TStringList.Duplicates` (285) setting has effect. This setting is ignored when Sorted is False.

See also: `TStringList.Sort` (285), `TStringList.Duplicates` (285), `TStringList.Add` (283), `TstringList.Insert` (285)

TStringList.OnChange

Synopsis: Event triggered after the list was modified.

Declaration: Property OnChange : TNotifyEvent

Visibility: public

Access: Read,Write

Description: OnChange can be assigned to respond to changes that have occurred in the list. The handler is called whenever strings are added, moved, modified or deleted from the list.

The Onchange event is triggered after the modification took place. When the modification is about to happen, an `TstringList.OnChanging` (286) event occurs.

See also: `TStringList.OnChanging` (286)

TStringList.OnChanging

Synopsis: Event triggered when the list is about to be modified.

Declaration: Property OnChanging : TNotifyEvent

Visibility: public

Access: Read,Write

Description: `OnChangeing` can be assigned to respond to changes that will occurred in the list. The handler is called whenever strings will be added, moved, modified or deleted from the list.

The `OnChangeing` event is triggered before the modification will take place. When the modification has happened, an `TstringList.OnChange` (286) event occurs.

See also: `TStringList.OnChange` (286)

2.50 TStrings

Description

`TStrings` implements an abstract class to manage an array of strings. It introduces methods to set and retrieve strings in the array, searching for a particular string, concatenating the strings and so on. It also allows an arbitrary object to be associated with each string.

It also introduces methods to manage a series of `name=value` settings, as found in many configuration files.

An instance of `TStrings` is never created directly, instead a descendent class such as `TStringList` (280) should be created. This is because `TStrings` is an abstract class which does not implement all methods; `TStrings` also doesn't store any strings, this is the functionality introduced in descendents such as `TStringList` (280).

Method overview

Page	Method	Description
292	Add	Add a string to the list
292	AddObject	Add a string and associated object to the list.
293	AddStrings	Add contents of another stringlist to this list.
293	Append	Add a string to the list.
293	Assign	Assign the contents of another stringlist to this one.
293	BeginUpdate	Mark the beginning of an update batch.
294	Clear	Removes all strings and associated objects from the list.
289	DefineProperties	Method to stream the contents of the string collection
294	Delete	Delete a string from the list.
292	Destroy	Frees all strings and objects, and removes the list from memory.
294	EndUpdate	Mark the end of an update batch.
295	Equals	Compares the contents of two stringlists.
289	Error	Raises an EStringListError (174) exception.
295	Exchange	Exchanges two strings in the list.
289	Get	Abstract read handler for the TStrings.Strings (302) property.
289	GetCapacity	Abstract Read handler for the TStrings.Capacity (299) property.
290	GetCount	Abstract read handler for the TStrings.Count (300) property.
290	GetObject	Abstract read handler for the TStrings.Objects (301) property.
295	GetText	Returns the contents as a PChar
290	GetTextStr	Read handler for the TStrings.Text (302) property.
295	IndexOf	Find a string in the list and return its position.
296	IndexOfName	Finds the index of a name in the name-value pairs.
296	IndexOfObject	Finds an object in the list and returns its index.
296	Insert	Insert a string in the list.
297	InsertObject	Insert a string and associated object in the list.
297	LoadFromFile	Load the contents of a file as a series of strings.
297	LoadFromStream	Load the contents of a stream as a series of strings.
298	Move	Move a string from one place in the list to another.
290	Put	Write handler for the TStrings.Strings (302) property.
291	PutObject	Write handler for the TStrings.Objects (301) property.
298	SaveToFile	Save the contents of the list to a file.
299	SaveToStream	Save the contents of the string to a stream.
291	SetCapacity	Write handler for the TStrings.Capacity (299) property.
299	SetText	Set the contents of the list from a PChar.
291	SetTextStr	Write handler for the TStrings.Text (302) property.
292	SetUpdateState	Sets the update state.

Property overview

Page	Property	Access	Description
299	Capacity	rw	Capacity of the list, i.e. number of strings that the list can currently hold before it tries to expand.
299	CommaText	rw	Contents of the list as a comma-separated string.
300	Count	r	Number of strings in the list.
300	Names	r	Name parts of the name-value pairs in the list.
301	Objects	rw	Indexed access to the objects associated with the strings in the list.
302	Strings	rw	Indexed access to teh strings in the list.
302	StringsAdapter	rw	Not implemented in Free Pascal.
302	Text	rw	Contents of the list as one big string.
301	Values	rw	Value parts of the name-value pairs in the list.

TStrings.DefineProperties

Synopsis: Method to stream the contents of the string collection

Declaration: `procedure DefineProperties(Filer: TFile); Override`

Visibility: protected

Description: `DefineProperties` allows the contents of the string collection to be streamed. As such, it overrides `TPersistent.DefineProperties` (256)

See also: `TPersistent.DefineProperties` (256)

TStrings.Error

Synopsis: Raises an `EStringListError` (174) exception.

Declaration: `procedure Error(const Msg: String; Data: Integer)`

Visibility: protected

Description: `Error` raises an `EStringListError` (174) exception. It passes `Msg` as a format with `Data` as the only argument.

This method can be used by descendent objects to raise an error.

See also: `EStringListError` (174)

TStrings.Get

Synopsis: Abstract read handler for the `TStrings.Strings` (302) property.

Declaration: `function Get(Index: Integer) : String; Virtual; Abstract`

Visibility: protected

Description: `Get` is the abstract read handler for the `TStrings.Strings` (302) property. This is an abstract method, hence it is not implemented in `TStrings`.

Descendent classes, such as `TStringList` (280) must override this method and implement a routine that retrieves the `Index`-th string in the list. `Index` should have a value between 0 and `Count-1`, in all other cases an error should be raised using `TStrings.Error` (289).

See also: `TStrings.Strings` (302), `TStrings.Put` (290), `TStrings.GetObject` (290)

TStrings.GetCapacity

Synopsis: Abstract Read handler for the `TStrings.Capacity` (299) property.

Declaration: `function GetCapacity : Integer; Virtual`

Visibility: protected

Description: `GetCapacity` is the read handler for the `TStrings.Capacity` (299) property. The implementation in `TStrings` will return 0.

Descendent classes can override this method. It should return the current number of strings that can be held by the stringlist before it attempts to expand it's storage space.

See also: `TStrings.Capacity` (299), `TStrings.SetCapacity` (291)

TStrings.GetCount

Synopsis: Abstract read handler for the TStrings.Count (300) property.

Declaration: `function GetCount : Integer; Virtual; Abstract`

Visibility: protected

Description: `GetCount` is the abstract read handler for the TStrings.Count (300) property. This is an abstract method, hence it is not implemented in TStrings.

Descendent classes must override this method. It should return the current number of strings in the list. (empty strings included).

See also: TStrings.Count (300)

TStrings.GetObject

Synopsis: Abstract read handler for the TStrings.Objects (301) property.

Declaration: `function GetObject(Index: Integer) : TObject; Virtual`

Visibility: protected

Description: `GetObject` is the read handler for the TStrings.Objects (301) property. The TStrings implementation of this method ignores the `Index` argument and simply returns `Nil`.

Descendent classes that should support object storage should override this method and return the object associated to the `Index`-th string in the list. `Index` should have a value between 0 and `Count-1`. If `Index` is outside the allowed range, an error should be raised using TStrings.Error (289).

See also: TStrings.Objects (301), TStrings.PutObject (291), TStrings.Get (289)

TStrings.GetTextStr

Synopsis: Read handler for the TStrings.Text (302) property.

Declaration: `function GetTextStr : String; Virtual`

Visibility: protected

Description: `GetTextStr` is the read handler for the TStrings.Text (302) property. It simply concatenates all strings in the list with a linefeed between them, and returns the resulting string.

Descendent classes may override this method to implement an efficient algorithm which is more suitable to their storage method.

See also: TStrings.Text (302), TStrings.SetTextStr (291)

TStrings.Put

Synopsis: Write handler for the TStrings.Strings (302) property.

Declaration: `procedure Put(Index: Integer; const S: String); Virtual`

Visibility: protected

Description: Put is the write handler for the TStrings.Strings (302) property. It does this by saving the object associated to the Index-th string, deleting the Index-th string, and inserting S and the saved object at position Index with TStrings.InsertObject (297)

Descendent classes may wish to override Put to implement a more efficient method.

See also: TStrings.Strings (302), TStrings.Get (289), TStrings.PutObject (291)

TStrings.PutObject

Synopsis: Write handler for the TStrings.Objects (301) property.

Declaration: `procedure PutObject(Index: Integer; AObject: TObject); Virtual`

Visibility: protected

Description: PutObject is the write handler for the TStrings.Objects (301) property. The TStrings implementation of PutObject does nothing.

Descendent objects that should support Object storage must override this method to store the AObject so that it is associated with the Index-th string in the list. Index should have a value between 0 and Count-1. If the value of Index is out of range, an error should be raised using TStrings.Error (289).

See also: TStrings.Objects (301), TStrings.GetObject (290), TStrings.Put (290)

TStrings.SetCapacity

Synopsis: Write handler for the TStrings.Capacity (299) property.

Declaration: `procedure SetCapacity(NewCapacity: Integer); Virtual`

Visibility: protected

Description: SetCapacity is the write handler for the TStrings.Capacity (299) property. The TStrings implementation of SetCapacity does nothing.

Descendent classes can override this method to set the current capacity of the stringlist to NewCapacity. The capacity is the number of strings the list can hold before it tries to expand its storage space. NewCapacity should be no less than Count.

See also: TStrings.Capacity (299), TStrings.GetCapacity (289)

TStrings.SetTextStr

Synopsis: Write handler for the TStrings.Text (302) property.

Declaration: `procedure SetTextStr(const Value: String); Virtual`

Visibility: protected

Description: SetTextStr is the write method for the TStrings.Text (302) property. It does nothing other than calling TStrings.SetText (299).

Descendent classes may override this method to implement a more efficient algorithm that fits their storage method better.

See also: TStrings.Text (302), TStrings.GetTextStr (290)

TStrings.SetUpdateState

Synopsis: Sets the update state.

Declaration: `procedure SetUpdateState(Updating: Boolean); Virtual`

Visibility: `protected`

Description: `SetUpdateState` sets the update state to `Updating`. The `TStrings` implementation of `SetUpdateState` does nothing.

Descendent objects may override this method to implement optimizations. If `Updating` is `True` then the list of strings is about to be updated (possibly many times). If it is `False` no more updates will take place till the next `SetUpdateState` call.

See also: `TStrings.BeginUpdate` ([293](#)), `TStrings.EndUpdate` ([294](#))

TStrings.Destroy

Synopsis: Frees all strings and objects, and removes the list from memory.

Declaration: `destructor Destroy; Override`

Visibility: `public`

Description: `Destroy` is the destructor of `TStrings` it does nothing except calling the inherited destructor.

TStrings.Add

Synopsis: Add a string to the list

Declaration: `function Add(const S: String) : Integer; Virtual`

Visibility: `public`

Description: `Add` adds `S` at the end of the list and returns the index of `S` in the list (which should equal `TStrings.Count` ([300](#)))

See also: `TStrings.Items` ([287](#)), `TStrings.AddObject` ([292](#)), `TStrings.Insert` ([296](#)), `TStrings.Delete` ([294](#)), `TStrings.Strings` ([302](#)), `TStrings.Count` ([300](#))

TStrings.AddObject

Synopsis: Add a string and associated object to the list.

Declaration: `function AddObject(const S: String; AObject: TObject) : Integer; Virtual`

Visibility: `public`

Description: `AddObject` adds `S` to the list of strings, and associates `AObject` with it. It returns the index of `S`.

Remark: An object added to the list is not automatically destroyed by the list of the list is destroyed or the string it is associated with is deleted. It is the responsibility of the application to destroy any objects associated with strings.

See also: `TStrings.Add` ([292](#)), `Tstrings.Items` ([287](#)), `TStrings.Objects` ([301](#)), `Tstrings.InsertObject` ([297](#))

TStrings.Append

Synopsis: Add a string to the list.

Declaration: `procedure Append(const S: String)`

Visibility: public

Description: `Append` does the same as `TStrings.Add` (292), only it does not return the index of the inserted string.

See also: `TStrings.Add` (292)

TStrings.AddStrings

Synopsis: Add contents of another stringlist to this list.

Declaration: `procedure AddStrings(TheStrings: TStrings); Virtual`

Visibility: public

Description: `AddStrings` adds the contents of `TheStrings` to the stringlist. Any associated objects are added as well.

See also: `TStrings.Add` (292), `TStrings.Assign` (293)

TStrings.Assign

Synopsis: Assign the contents of another stringlist to this one.

Declaration: `procedure Assign(Source: TPersistent); Override`

Visibility: public

Description: `Assign` replaces the contents of the stringlist with the contents of `Source` if `Source` is also of type `TStrings`. Any associated objects are copied as well.

See also: `TStrings.Add` (292), `TStrings.AddStrings` (293), `TPersistent.Assign` (257)

TStrings.BeginUpdate

Synopsis: Mark the beginning of an update batch.

Declaration: `procedure BeginUpdate`

Visibility: public

Description: `BeginUpdate` increases the update count by one. It is advisable to call `BeginUpdate` before lengthy operations on the stringlist. At the end of these operation, `TStrings.EndUpdate` (294) should be called to mark the end of the operation. Descendent classes may use this information to perform optimizations. e.g. updating the screen only once after many strings were added to the list.

All `TStrings` methods that modify the string list call `BeginUpdate` before the actual operation, and call `endUpdate` when the operation is finished. Descendent classes should also call these methods when modifying the string list.

Remark: Always put the corresponding call to `TStrings.EndUpdate` (294) in the context of a `Finally` block, to ensure that the update count is always decreased at the end of the operation, even if an exception occurred:

```

With MyStrings do
  try
    BeginUpdate;
    // Some lengthy operation.
  Finally
    EndUpdate
  end;

```

See also: [TStrings.EndUpdate \(294\)](#)

TStrings.Clear

Synopsis: Removes all strings and associated objects from the list.

Declaration: `procedure Clear; Virtual; Abstract`

Visibility: public

Description: `Clear` will remove all strings and their associated objects from the list. After a call to `clear`, `TStrings.Count (300)` is zero.

Since it is an abstract method, `TStrings` itself does not implement `Clear`. Descendent classes such as `TStringList (280)` implement this method.

See also: [TStrings.Items \(287\)](#), [TStrings.Delete \(294\)](#), [TStrings.Count \(300\)](#)

TStrings.Delete

Synopsis: Delete a string from the list.

Declaration: `procedure Delete(Index: Integer); Virtual; Abstract`

Visibility: public

Description: `Delete` deletes the string at position `Index` from the list. The associated object is also removed from the list, but not destroyed. `Index` is zero-based, and should be in the range 0 to `Count-1`.

Since it is an abstract method, `TStrings` itself does not implement `Delete`. Descendent classes such as `TStringList (280)` implement this method.

Errors: If `Index` is not in the allowed range, an `EStringListError (174)` is raised.

See also: [TStrings.Insert \(296\)](#), [TStrings.Items \(287\)](#), [TStrings.Clear \(294\)](#)

TStrings.EndUpdate

Synopsis: Mark the end of an update batch.

Declaration: `procedure EndUpdate`

Visibility: public

Description: `EndUpdate` should be called at the end of a lengthy operation on the stringlist, but only if there was a call to `BeginUpdate` before the operation was started. It is best to put the call to `EndUpdate` in the context of a `Finally` block, so it will be called even if an exception occurs.

For more information, see [TStrings.BeginUpdate \(293\)](#).

See also: [TStrings.BeginUpdate \(293\)](#)

TStrings.Equals

Synopsis: Compares the contents of two stringlists.

Declaration: `function Equals(TheStrings: TStrings) : Boolean`

Visibility: public

Description: `Equals` compares the contents of the stringlist with the contents of `TheStrings`. If the contents match, i.e. the stringlist contain an equal amount of strings, and all strings match, then `True` is returned. If the number of strings in the lists is unequal, or they contain one or more different strings, `False` is returned.

Remark:

- 1.The strings are compared case-insensitively.
- 2.The associated objects are not compared

See also: `TStrings.Items` ([287](#)), `TStrings.Count` ([300](#)), `TStrings.Assign` ([293](#))

TStrings.Exchange

Synopsis: Exchanges two strings in the list.

Declaration: `procedure Exchange(Index1: Integer; Index2: Integer); Virtual`

Visibility: public

Description: `Exchange` exchanges the strings at positions `Index1` and `Index2`. The associated objects are also exchanged.

Both indexes must be in the range of valid indexes, i.e. must have a value between 0 and `Count-1`.

Errors: If either `Index1` or `Index2` is not in the range of valid indexes, an `EStringListError` ([174](#)) exception is raised.

See also: `TStrings.Move` ([298](#)), `TStrings.Strings` ([302](#)), `TStrings.Count` ([300](#))

TStrings.GetText

Synopsis: Returns the contents as a `PChar`

Declaration: `function GetText : PChar; Virtual`

Visibility: public

Description: `GetText` allocates a memory buffer and compies the contents of the stringlist to this buffer as a series of strings, separated by an end-of-line marker. The buffer is zero terminated.

Remark: The caller is responsible for freeing the returned memory buffer.

TStrings.IndexOf

Synopsis: Find a string in the list and return its position.

Declaration: `function IndexOf(const S: String) : Integer; Virtual`

Visibility: public

Description: `IndexOf` searches the list for `S`. The search is case-insensitive. If a matching entry is found, its position is returned. If no matching string is found, `-1` is returned.

Remark:

1. Only the first occurrence of the string is returned.
2. The returned position is zero-based, i.e. 0 indicates the first string in the list.

See also: `TStrings.IndexOfObject` (296), `TStrings.IndexOfName` (296), `TStrings.Strings` (302)

TStrings.IndexOfName

Synopsis: Finds the index of a name in the name-value pairs.

Declaration: `function IndexOfName(const Name: String) : Integer`

Visibility: public

Description: `IndexOfName` searches in the list of strings for a name-value pair with name part `Name`. If such a pair is found, it returns the index of the pair in the stringlist. If no such pair is found, the function returns `-1`. The search is done case-insensitive.

Remark:

1. Only the first occurrence of a matching name-value pair is returned.
2. The returned position is zero-based, i.e. 0 indicates the first string in the list.

See also: `TStrings.IndexOf` (295), `TStrings.IndexOfObject` (296), `TStrings.Strings` (302)

TStrings.IndexOfObject

Synopsis: Finds an object in the list and returns its index.

Declaration: `function IndexOfObject(AObject: TObject) : Integer`

Visibility: public

Description: `IndexOfObject` searches through the list of strings till it find a string associated with `AObject`, and returns the index of this string. If no such string is found, `-1` is returned.

Remark:

1. Only the first occurrence of a string with associated object `AObject` is returned; if more strings in the list can be associated with `AObject`, they will not be found by this routine.
2. The returned position is zero-based, i.e. 0 indicates the first string in the list.

TStrings.Insert

Synopsis: Insert a string in the list.

Declaration: `procedure Insert(Index: Integer; const S: String); Virtual; Abstract`

Visibility: public

Description: `Insert` inserts the string `S` at position `Index` in the list. `Index` is a zero-based position, and can have values from 0 to `Count`. If `Index` equals `Count` then the string is appended to the list.

Remark:

1. All methods that add strings to the list use `Insert` to add a string to the list.
2. If the string has an associated object, use `TStrings.InsertObject` ([297](#)) instead.

Errors: If `Index` is less than zero or larger than `Count` then an `EStringListError` ([174](#)) exception is raised.

See also: `TStrings.Add` ([292](#)), `TStrings.InsertObject` ([297](#)), `TStrings.Append` ([293](#)), `TStrings.Delete` ([294](#))

TStrings.InsertObject

Synopsis: Insert a string and associated object in the list.

Declaration: `procedure InsertObject(Index: Integer; const S: String; AObject: TObject)`

Visibility: public

Description: `InsertObject` inserts the string `S` and its associated object `AObject` at position `Index` in the list. `Index` is a zero-based position, and can have values from 0 to `Count`. If `Index` equals `Count` then the string is appended to the list.

Errors: If `Index` is less than zero or larger than `Count` then an `EStringListError` ([174](#)) exception is raised.

See also: `TStrings.Insert` ([296](#)), `TStrings.AddObject` ([292](#)), `TStrings.Append` ([293](#)), `TStrings.Delete` ([294](#))

TStrings.LoadFromFile

Synopsis: Load the contents of a file as a series of strings.

Declaration: `procedure LoadFromFile(const FileName: String); Virtual`

Visibility: public

Description: `LoadFromFile` loads the contents of a file into the stringlist. Each line in the file (as marked by the end-of-line marker of the particular OS the application runs on) becomes one string in the stringlist. This action replaces the contents of the stringlist, it does not append the strings to the current content.

`LoadFromFile` simply creates a file stream ([238](#)) with the given filename, and then executes `TStrings.LoadFromStream` ([297](#)); after that the file stream object is destroyed again.

See also: `TStrings.LoadFromStream` ([297](#)), `TStrings.SaveToFile` ([298](#)), `TStrings.SaveToStream` ([299](#))

TStrings.LoadFromStream

Synopsis: Load the contents of a stream as a series of strings.

Declaration: `procedure LoadFromStream(Stream: TStream); Virtual`

Visibility: public

Description: `LoadFromStream` loads the contents of `Stream` into the stringlist. Each line in the stream (as marked by the end-of-line marker of the particular OS the application runs on) becomes one string in the stringlist. This action replaces the contents of the stringlist, it does not append the strings to the current content.

See also: `TStrings.LoadFromFile` ([297](#)), `TStrings.SaveToFile` ([298](#)), `TStrings.SaveToStream` ([299](#))

TStrings.Move

Synopsis: Move a string from one place in the list to another.

Declaration: `procedure Move(CurIndex: Integer; NewIndex: Integer); Virtual`

Visibility: public

Description: Move moves the string at position `CurIndex` so it has position `NewIndex` after the move operation. The object associated to the string is also moved. `CurIndex` and `NewIndex` should be in the range of 0 to `Count-1`.

Remark: `NewIndex` is *not* the position in the stringlist before the move operation starts. The move operation

1. removes the string from position `CurIndex`
2. inserts the string at position `NewIndex`

This may not lead to the desired result if `NewIndex` is bigger than `CurIndex`. Consider the following example:

```
With MyStrings do
begin
  Clear;
  Add('String 0');
  Add('String 1');
  Add('String 2');
  Add('String 3');
  Add('String 4');
  Move(1,3);
end;
```

After the Move operation has completed, 'String 1' will be between 'String 3' and 'String 4'.

Errors: If either `CurIndex` or `NewIndex` is outside the allowed range, an `EStringListError` (174) is raised.

See also: `TStrings.Exchange` (295)

TStrings.SaveToFile

Synopsis: Save the contents of the list to a file.

Declaration: `procedure SaveToFile(const FileName: String); Virtual`

Visibility: public

Description: `SaveToFile` saves the contents of the stringlist to the file with name `FileName`. It writes the strings to the file, separated by end-of-line markers, so each line in the file will contain 1 string from the stringlist.

`SaveToFile` creates a file stream (238) with name `FileName`, calls `TStrings.SaveToStream` (299) and then destroys the file stream object.

Errors: An `EStreamError` (173) exception can be raised if the file `FileName` cannot be opened, or if it cannot be written to.

See also: `TStrings.SaveToStream` (299), `Tstrings.LoadFromStream` (297), `TStrings.LoadFromFile` (297)

TStrings.SaveToStream

Synopsis: Save the contents of the string to a stream.

Declaration: `procedure SaveToStream(Stream: TStream); Virtual`

Visibility: public

Description: `SaveToStream` saves the contents of the stringlist to `Stream`. It writes the strings to the stream, separated by end-of-line markers, so each 'line' in the stream will contain 1 string from the stringlist.

Errors: An `EStreamError` ([173](#)) exception can be raised if the stream cannot be written to.

See also: `TStrings.SaveToFile` ([298](#)), `Tstrings.LoadFromStream` ([297](#)), `TStrings.LoadFromFile` ([297](#))

TStrings.SetText

Synopsis: Set the contents of the list from a `PChar`.

Declaration: `procedure SetText(TheText: PChar); Virtual`

Visibility: public

Description: `SetText` parses the contents of `TheText` and fills the stringlist based on the contents. It regards `TheText` as a series of strings, separated by end-of-line markers. Each of these strings is added to the stringlist.

See also: `TStrings.Text` ([302](#))

TStrings.Capacity

Synopsis: Capacity of the list, i.e. number of strings that the list can currently hold before it tries to expand.

Declaration: `Property Capacity : Integer`

Visibility: public

Access: Read,Write

Description: `Capacity` is the number of strings that the list can hold before it tries to allocate more memory.

`TStrings` returns `TStrings.Count` ([300](#)) when read. Trying to set the capacity has no effect. Descendent classes such as `TStringList` ([280](#)) can override this property such that it actually sets the new capacity.

See also: `TStringList` ([280](#)), `TStrings.Count` ([300](#))

TStrings.CommaText

Synopsis: Contents of the list as a comma-separated string.

Declaration: `Property CommaText : String`

Visibility: public

Access: Read,Write

Description: `CommaText` represents the stringlist as a single string, consisting of a comma-separated concatenation of the strings in the list. If one of the strings contains spaces, comma's or quotes it will be enclosed by double quotes. Any double quotes in a string will be doubled. For instance the following strings:


```
Comma,string
Quote"string
Space string
NormalSttring
```

is converted to

```
"Comma,string","Quote"String","Space string",NormalString
```

Conversely, when setting the `CommaText` property, the text will be parsed according to the rules outlined above, and the strings will be set accordingly. Note that spaces will in this context be regarded as string separators, unless the string as a whole is contained in double quotes. Spaces that occur next to a delimiter will be ignored. The following string:

```
"Comma,string" , "Quote"String",Space string,, NormalString
```

Will be converted to

```
Comma,String
Quote"String
Space
String

NormalString
```

See also: `TStrings.Text` ([302](#)), `TStrings.SetText` ([299](#))

TStrings.Count

Synopsis: Number of strings in the list.

Declaration: `Property Count : Integer`

Visibility: `public`

Access: `Read`

Description: `Count` is the current number of strings in the list. `TStrings` does not implement this property; descendent classes should override the property read handler to return the correct value.

Strings in the list are always uniquely identified by their `Index`; the index of a string is zero-based, i.e. it's supported range is 0 to `Count-1`. trying to access a string with an index larger than or equal to `Count` will result in an error. Code that iterates over the list in a stringlist should always take into account the zero-based character of the list index.

See also: `TStrings.Strings` ([302](#)), `TStrings.Objects` ([301](#)), `TStrings.Capacity` ([299](#))

TStrings.Names

Synopsis: Name parts of the name-value pairs in the list.

Declaration: `Property Names[Index: Integer]: String`

Visibility: `public`

Access: Read

Description: Names provides indexed access to the names of the name-value pairs in the list. It returns the name part of the `Index`-th string in the list.

Remark: The index is not an index based on the number of name-value pairs in the list. It is the name part of the name-value pair a string `Index` in the list. If the string at position `Index` is not a name-value pair (i.e. does not contain the equal sign (=)), then an empty name is returned.

See also: `TStrings.Values` (301), `TStrings.IndexOfName` (296)

TStrings.Objects

Synopsis: Indexed access to the objects associated with the strings in the list.

Declaration: `Property Objects[Index: Integer]: TObject`

Visibility: public

Access: Read,Write

Description: `Objects` provides indexed access to the objects associated to the strings in the list. `Index` is a zero-based index and must be in the range of 0 to `Count-1`.

Setting the `objects` property will not free the previously associated object, if there was one. The caller is responsible for freeing the object that was previously associated to the string.

`TStrings` does not implement any storage for objects. Reading the `Objects` property will always return `Nil`. Setting the property will have no effect. It is the responsibility of the descendent classes to provide storage for the associated objects.

Errors: If an `Index` outside the valid range is specified, an `EStringListError` (174) exception will be raised.

See also: `TStrings.Strings` (302), `TStrings.IndexOfObject` (296), `TStrings.Names` (300), `TStrings.Values` (301)

TStrings.Values

Synopsis: Value parts of the name-value pairs in the list.

Declaration: `Property Values[Name: String]: String`

Visibility: public

Access: Read,Write

Description: `Values` represents the value parts of the name-value pairs in the list.

When reading this property, if there is a name-value pair in the list of strings that has name part `Name`, then the corresponding value is returned. If there is no such pair, an empty string is returned.

When writing this value, first it is checked whether there exists a name-value pair in the list with name `Name`. If such a pair is found, its value part is overwritten with the specified value. If no such pair is found, a new name-value pair is added with the specified `Name` and value.

Remark:

1. Names are compared case-insensitively.
2. Any character, including whitespace, up till the first equal (=) sign in a string is considered part of the name.

See also: `TStrings.Names` (300), `TStrings.Strings` (302), `TStrings.Objects` (301)

TStrings.Strings

Synopsis: Indexed access to the strings in the list.

Declaration: Property Strings[Index: Integer]: String; default

Visibility: public

Access: Read,Write

Description: Strings is the default property of TStrings. It provides indexed read-write access to the list of strings. Reading it will return the string at position Index in the list. Writing it will set the string at position Index.

Index is the position of the string in the list. It is zero-based, i.e. valid values range from 0 (the first string in the list) till Count-1 (the last string in the list). When browsing through the strings in the list, this fact must be taken into account.

To access the objects associated with the strings in the list, use the TStrings.Objects (301) property. The name parts of name-value pairs can be accessed with the TStrings.Names (300) property, and the values can be set or read through the TStrings.Values (301) property.

Searching through the list can be done using the TStrings.IndexOf (295) method.

Errors: If Index is outside the allowed range, an EStringListError (174) exception is raised.

See also: TStrings.Count (300), TStrings.Objects (301), TStrings.Names (300), TStrings.Values (301), TStrings.IndexOf (295)

TStrings.Text

Synopsis: Contents of the list as one big string.

Declaration: Property Text : String

Visibility: public

Access: Read,Write

Description: Text returns, when read, the contents of the stringlist as one big string consisting of all strings in the list, separated by an end-of-line marker. When this property is set, the string will be cut into smaller strings, based on the positions of end-of-line markers in the string. Any previous content of the stringlist will be lost.

Remark: If any of the strings in the list contains an end-of-line marker, then the resulting string will appear to contain more strings than actually present in the list. To avoid this ambiguity, use the TStrings.CommaText (299) property instead.

See also: TStrings.Strings (302), TStrings.Count (300), TStrings.CommaText (299)

TStrings.StringsAdapter

Synopsis: Not implemented in Free Pascal.

Declaration: Property StringsAdapter : IStringsAdapter

Visibility: public

Access: Read,Write

Description: Not implemented in Free Pascal.

2.51 TStringStream

Description

TStringStream stores its data in an ansistring. The contents of this string is available as the `DataStream` (305) property. It also introduces some methods to read or write parts of the stringstream's data as a string.

The main purpose of a TStringStream is to be able to treat a string as a stream from which can be read.

Method overview

Page	Method	Description
303	Create	Creates a new stringstream and sets its initial content.
304	Read	Reads from the stream.
304	ReadString	Reads a string of length Count
304	Seek	Sets the position in the stream.
303	SetSize	Sets the size of the stream.
304	Write	Write implements the abstract TStream.Write (273) method.
304	WriteString	WriteString writes a string to the stream.

Property overview

Page	Property	Access	Description
305	DataStream	r	Contains the contents of the stream in string form

TStringStream.SetSize

Synopsis: Sets the size of the stream.

Declaration: `procedure SetSize(NewSize: LongInt); Override`

Visibility: `protected`

Description: `SetSize` sets the size of the stream to `newsize`. It does this by setting the size of the ansistring in which the stream is stored. `NewSize` can have any value greater than or equal to zero.

Errors: In case there is not enough memory, an exception may be raised.

See also: `TStream.Size` (280)

TStringStream.Create

Synopsis: Creates a new stringstream and sets its initial content.

Declaration: `constructor Create(const AString: String)`

Visibility: `public`

Description: `Create` creates a new `TStringStream` instance and sets its initial content to `AString`. The position is still 0 but the size of the stream will equal the length of the string.

See also: `TStringStream.DataString` (305)

TStringStream.Read

Synopsis: Reads from the stream.

Declaration: `function Read(var Buffer;Count: LongInt) : LongInt; Override`

Visibility: public

Description: Read implements the abstract Read (272) from TStream (271). It tries to read Count bytes into Buffer. It returns the number of bytes actually read. The position of the stream is advanced with the number of bytes actually read; When the reading has reached the end of the DataString (305), then the reading stops, i.e. it is not possible to read beyond the end of the datastring.

See also: TStream.Read (272), TStringStream.Write (304), TStringStream.DataString (305)

TStringStream.ReadString

Synopsis: Reads a string of length Count

Declaration: `function ReadString(Count: LongInt) : String`

Visibility: public

Description: ReadString reads Count bytes from the stream and returns the read bytes as a string. If less than Count bytes were available, the string has as many characters as bytes could be read.

The ReadString method is a wrapper around the Read (304) method. It does not do the same stringas the TStream.ReadAnsiString (278) method, which first reads a length integer to determine the length of the string to be read.

See also: TStringStream.Read (304), TStream.ReadAnsiString (278)

TStringStream.Seek

Synopsis: Sets the position in the stream.

Declaration: `function Seek(Offset: LongInt;Origin: Word) : LongInt; Override`

Visibility: public

Description: Seek implements the abstract Seek (273) method.

TStringStream.Write

Synopsis: Write implements the abstract TStream.Write (273) method.

Declaration: `function Write(const Buffer;Count: LongInt) : LongInt; Override`

Visibility: public

Description: Write implements the abstract TStream.Write (273) method.

TStringStream.WriteString

Synopsis: WriteString writes a string to the stream.

Declaration: `procedure WriteString(const AString: String)`

Visibility: public

Description: WriteString writes a string to the stream.

TStringStream.DataString

Synopsis: Contains the contents of the stream in string form

Declaration: Property DataString : String

Visibility: public

Access: Read

Description: Contains the contents of the stream in string form

2.52 TTextWriter**Description**

Not yet implemented.

2.53 TThread**Description**

The `TThread` class encapsulates the native thread support of the operating system. To create a thread, declare a descendent of the `TThread` object and override the `Execute` (306) method. In this method, the thread's code should be executed. To run a thread, create an instance of the thread descendent, and call its `execute` method.

Method overview

Page	Method	Description
306	Create	Creates a new thread.
306	Destroy	Destroys the thread object.
306	DoTerminate	Terminates the thread.
306	Execute	Execute method. Should be overridden in a descendent thread.
306	Resume	Resumes the thread's execution.
307	Suspend	Suspends the thread's execution.
306	Synchronize	Synchronizes the thread by executing the method in the main thread.
307	Terminate	Signals the thread it should terminate.
307	WaitFor	Waits for the thread to terminate and returns the exit status.

Property overview

Page	Property	Access	Description
309	FatalException	r	
308	FreeOnTerminate	rw	Indicates whether the thread should free itself when it stops executing.
308	Handle	r	Returns the thread handle.
309	OnTerminate	rw	Event called when the thread terminates.
308	Priority	rw	Returns the thread priority.
307	ReturnValue	rw	Return value of the thread when it stops executing.
308	Suspended	rw	Indicates whether the thread is suspended.
307	Terminated	r	Indicates whether the <code>Terminate</code> (307) method was called by the user.
308	ThreadID	r	Returns the thread ID.

TThread.DoTerminate

Synopsis: Terminates the thread.

Declaration: `procedure DoTerminate; Virtual`

Visibility: `protected`

Description: Terminates the thread.

TThread.Execute

Synopsis: Execute method. Should be overridden in a descendent thread.

Declaration: `procedure Execute; Virtual; Abstract`

Visibility: `protected`

Description: Execute method. Should be overridden in a descendent thread.

TThread.Synchronize

Synopsis: Synchronizes the thread by executing the method in the main thread.

Declaration: `procedure Synchronize(Method: TThreadMethod)`

Visibility: `protected`

Description: Synchronizes the thread by executing the method in the main thread.

TThread.Create

Synopsis: Creates a new thread.

Declaration: `constructor Create(CreateSuspended: Boolean)`

Visibility: `public`

Description: Creates a new thread.

TThread.Destroy

Synopsis: Destroys the thread object.

Declaration: `destructor Destroy; Override`

Visibility: `public`

Description: Destroys the thread object.

TThread.Resume

Synopsis: Resumes the thread's execution.

Declaration: `procedure Resume`

Visibility: `public`

Description: Resumes the thread's execution.

TThread.Suspend

Synopsis: Suspends the thread's execution.

Declaration: `procedure Suspend`

Visibility: `public`

Description: Suspends the thread's execution.

TThread.Terminate

Synopsis: Signals the thread it should terminate.

Declaration: `procedure Terminate`

Visibility: `public`

Description: Signals the thread it should terminate.

TThread.WaitFor

Synopsis: Waits for the thread to terminate and returns the exit status.

Declaration: `function WaitFor : Integer`

Visibility: `public`

Description: Waits for the thread to terminate and returns the exit status.

TThread.ReturnValue

Synopsis: Return value of the thread when it stops executing.

Declaration: `Property ReturnValue : Integer`

Visibility: `protected`

Access: `Read,Write`

Description: Return value of the thread when it stops executing.

TThread.Terminated

Synopsis: Indicates whether the `Terminate` (307) method was called by the user.

Declaration: `Property Terminated : Boolean`

Visibility: `protected`

Access: `Read`

Description: Indicates whether the `Terminate` (307) method was called by the user.

TThread.FreeOnTerminate

Synopsis: Indicates whether the thread should free itself when it stops executing.

Declaration: `Property FreeOnTerminate : Boolean`

Visibility: `public`

Access: `Read,Write`

Description: Indicates whether the thread should free itself when it stops executing.

TThread.Handle

Synopsis: Returns the thread handle.

Declaration: `Property Handle : THandle`

Visibility: `public`

Access: `Read`

Description: Returns the thread handle.

TThread.Priority

Synopsis: Returns the thread priority.

Declaration: `Property Priority : TThreadPriority`

Visibility: `public`

Access: `Read,Write`

Description: Returns the thread priority.

TThread.Suspended

Synopsis: Indicates whether the thread is suspended.

Declaration: `Property Suspended : Boolean`

Visibility: `public`

Access: `Read,Write`

Description: Indicates whether the thread is suspended.

TThread.ThreadID

Synopsis: Returns the thread ID.

Declaration: `Property ThreadID : THandle`

Visibility: `public`

Access: `Read`

Description: Returns the thread ID.

TThread.OnTerminate

Synopsis: Event called when the thread terminates.

Declaration: `Property OnTerminate : TNotifyEvent`

Visibility: `public`

Access: `Read,Write`

Description: Event called when the thread terminates.

TThread.FatalException

Declaration: `Property FatalException : TObject`

Visibility: `public`

Access: `Read`

2.54 TThreadList

Description

This class is not yet implemented in Free Pascal.

Method overview

Page	Method	Description
310	Add	Adds an element to the list.
310	Clear	Removes all emements from the list.
309	Create	Creates a new thread-safe list.
309	Destroy	Destroys the list instance.
310	LockList	Locks the list for exclusive access.
310	Remove	Removes an item from the list.
311	UnlockList	Unlocks the list after it was locked.

TThreadList.Create

Synopsis: Creates a new thread-safe list.

Declaration: `constructor Create`

Visibility: `public`

Description: This class is not yet implemented in Free Pascal.

Errors:

TThreadList.Destroy

Synopsis: Destroys the list instance.

Declaration: `destructor Destroy; Override`

Visibility: `public`

Description: This class is not yet implemented in Free Pascal.

Errors:

TThreadList.Add

Synopsis: Adds an element to the list.

Declaration: `procedure Add(Item: Pointer)`

Visibility: `public`

Description: This class is not yet implemented in Free Pascal.

Errors:

TThreadList.Clear

Synopsis: Removes all elements from the list.

Declaration: `procedure Clear`

Visibility: `public`

Description: This class is not yet implemented in Free Pascal.

Errors:

TThreadList.LockList

Synopsis: Locks the list for exclusive access.

Declaration: `function LockList : TList`

Visibility: `public`

Description: This class is not yet implemented in Free Pascal.

Errors:

TThreadList.Remove

Synopsis: Removes an item from the list.

Declaration: `procedure Remove(Item: Pointer)`

Visibility: `public`

Description: This class is not yet implemented in Free Pascal.

Errors:

TThreadList.UnlockList

Synopsis: Unlocks the list after it was locked.

Declaration: `procedure UnlockList`

Visibility: `public`

Description: This class is not yet implemented in Free Pascal.

Errors:

2.55 TWriter**Description**

Object to write component data to an arbitrary format.

Method overview

Page	Method	Description
312	Create	Creates a new Writer with a stream and bufsize.
313	DefineBinaryProperty	Callback used when defining and streaming custom properties.
313	DefineProperty	Callback used when defining and streaming custom properties.
312	Destroy	Destroys the writer instance.
312	SetRoot	Sets the root component
312	WriteBinary	Writes binary data to the stream.
313	WriteBoolean	Write boolean value to the stream.
314	WriteChar	Write a character to the stream.
313	WriteCollection	Write a collection to the stream.
313	WriteComponent	Stream a component to the stream.
314	WriteDate	Write a date to the stream.
314	WriteDescendent	Write a descendent component to the stream.
314	WriteFloat	Write a float to the stream.
314	WriteIdent	Write an identifier to the stream.
315	WriteInteger	Write an integer to the stream.
315	WriteListBegin	Write a start-of-list marker to the stream.
315	WriteListEnd	Write an end-of-list marker to the stream.
312	WriteProperties	Writes the published properties to the stream.
312	WriteProperty	Writes one property to the stream.
315	WriteRootComponent	Write a root component to the stream.
314	WriteSingle	Write a single-type real to the stream.
315	WriteString	Write a string to the stream.

Property overview

Page	Property	Access	Description
316	Driver	r	Driver used when writing to the stream.
316	OnFindAncestor	rw	Event occurring when an ancestor component must be found.
316	OnWriteMethodProperty	rw	
316	OnWriteStringProperty	rw	
316	RootAncestor	rw	Ancestor of root component.

TWriter.SetRoot

Synopsis: Sets the root component

Declaration: `procedure SetRoot(ARoot: TComponent); Override`

Visibility: `protected`

TWriter.WriteBinary

Synopsis: Writes binary data to the stream.

Declaration: `procedure WriteBinary(AWriteData: TStreamProc)`

Visibility: `protected`

Description: Writes binary data to the stream.

TWriter.WriteProperty

Synopsis: Writes one property to the stream.

Declaration: `procedure WriteProperty(Instance: TPersistent; PropInfo: Pointer)`

Visibility: `protected`

Description: Writes one property to the stream.

TWriter.WriteProperties

Synopsis: Writes the published properties to the stream.

Declaration: `procedure WriteProperties(Instance: TPersistent)`

Visibility: `protected`

Description: Writes the published properties to the stream.

TWriter.Create

Synopsis: Creates a new Writer with a stream and bufsize.

Declaration: `constructor Create(ADriver: TAbstractObjectWriter)`
`constructor Create(Stream: TStream; BufSize: Integer)`

Visibility: `public`

Description: Creates a new Writer with a stream and bufsize.

TWriter.Destroy

Synopsis: Destroys the writer instance.

Declaration: `destructor Destroy; Override`

Visibility: `public`

Description: Destroys the writer instance.

TWriter.DefineProperty

Synopsis: Callback used when defining and streaming custom properties.

Declaration:

```
procedure DefineProperty(const Name: String; ReadData: TReaderProc;
                        AWriteData: TWriterProc; HasData: Boolean)
; Override
```

Visibility: public

Description: Callback used when defining and streaming custom properties.

TWriter.DefineBinaryProperty

Synopsis: Callback used when defining and streaming custom properties.

Declaration:

```
procedure DefineBinaryProperty(const Name: String; ReadData: TStreamProc;
                              AWriteData: TStreamProc; HasData: Boolean)
; Override
```

Visibility: public

Description: Callback used when defining and streaming custom properties.

TWriter.WriteBoolean

Synopsis: Write boolean value to the stream.

Declaration:

```
procedure WriteBoolean(Value: Boolean)
```

Visibility: public

Description: Write boolean value to the stream.

TWriter.WriteCollection

Synopsis: Write a collection to the stream.

Declaration:

```
procedure WriteCollection(Value: TCollection)
```

Visibility: public

Description: Write a collection to the stream.

TWriter.WriteComponent

Synopsis: Stream a component to the stream.

Declaration:

```
procedure WriteComponent(Component: TComponent)
```

Visibility: public

Description: Stream a component to the stream.

TWriter.WriteChar

Synopsis: Write a character to the stream.

Declaration: `procedure WriteChar(Value: Char)`

Visibility: `public`

Description: Write a character to the stream.

TWriter.WriteDescendent

Synopsis: Write a descendent component to the stream.

Declaration: `procedure WriteDescendent(ARoot: TComponent; AAncestor: TComponent)`

Visibility: `public`

Description: Write a descendent component to the stream.

TWriter.WriteFloat

Synopsis: Write a float to the stream.

Declaration: `procedure WriteFloat(const Value: Extended)`

Visibility: `public`

Description: Write a float to the stream.

TWriter.WriteSingle

Synopsis: Write a single-type real to the stream.

Declaration: `procedure WriteSingle(const Value: Single)`

Visibility: `public`

Description: Write a single-type real to the stream.

TWriter.WriteDate

Synopsis: Write a date to the stream.

Declaration: `procedure WriteDate(const Value: TDateTime)`

Visibility: `public`

Description: Write a date to the stream.

TWriter.WriteIdent

Synopsis: Write an identifier to the stream.

Declaration: `procedure WriteIdent(const Ident: String)`

Visibility: `public`

Description: Write an identifier to the stream.

TWriter.WriteInteger

Synopsis: Write an integer to the stream.

Declaration: `procedure WriteInteger(Value: LongInt); Overload`
`procedure WriteInteger(Value: Int64); Overload`

Visibility: `public`

Description: Write an integer to the stream.

TWriter.WriteListBegin

Synopsis: Write a start-of-list marker to the stream.

Declaration: `procedure WriteListBegin`

Visibility: `public`

Description: Write a start-of-list marker to the stream.

TWriter.WriteListEnd

Synopsis: Write an end-of-list marker to the stream.

Declaration: `procedure WriteListEnd`

Visibility: `public`

Description: Write an end-of-list marker to the stream.

TWriter.WriteRootComponent

Synopsis: Write a root component to the stream.

Declaration: `procedure WriteRootComponent(ARoot: TComponent)`

Visibility: `public`

Description: Write a root component to the stream.

TWriter.WriteString

Synopsis: Write a string to the stream.

Declaration: `procedure WriteString(const Value: String)`

Visibility: `public`

Description: Write a string to the stream.

TWriter.RootAncestor

Synopsis: Ancestor of root component.

Declaration: Property RootAncestor : TComponent

Visibility: public

Access: Read,Write

Description: Ancestor of root component.

TWriter.OnFindAncestor

Synopsis: Event occurring when an ancestor component must be found.

Declaration: Property OnFindAncestor : TFindAncestorEvent

Visibility: public

Access: Read,Write

Description: Event occurring when an ancestor component must be found.

TWriter.OnWriteMethodProperty

Declaration: Property OnWriteMethodProperty : TWriteMethodPropertyEvent

Visibility: public

Access: Read,Write

TWriter.OnWriteStringProperty

Declaration: Property OnWriteStringProperty : TReadWriteStringPropertyEvent

Visibility: public

Access: Read,Write

TWriter.Driver

Synopsis: Driver used when writing to the stream.

Declaration: Property Driver : TAbstractObjectWriter

Visibility: public

Access: Read

Description: Driver used when writing to the stream.

Chapter 3

Reference for unit 'Crt'

3.1 Overview

This chapter describes the CRT unit for Free Pascal, both under dos linux and Windows. The unit was first written for dos by Florian klaempfl. The unit was ported to linux by Mark May and enhanced by Michael Van Canneyt and Peter Vreman. It works on the linux console, and in xterm and rxvt windows under X-Windows. The functionality for both is the same, except that under linux the use of an early implementation (versions 0.9.1 and earlier of the compiler) the crt unit automatically cleared the screen at program startup.

There are some caveats when using the CRT unit:

- Programs using the CRT unit will *not* be usable when input/output is being redirected on the command-line.
- For similar reasons they are not usable as CGI-scripts for use with a webserver.
- The use of the CRT unit and the graph unit may not always be supported.
- On linux or other unix OSes , executing other programs that expect special terminal behaviour (using one of the special functions in the linux unit) will not work. The terminal is set in RAW mode, which will destroy most terminal emulation settings.

3.2 Constants, types and variables

Constants

Black = 0

Black color attribute

Blink = 128

Blink attribute

Blue = 1

Blue color attribute

Brown = 6

Brown color attribute

BW40 = 0

40 columns black and white screen mode.

BW80 = 2

80 columns black and white screen mode.

C40 = CO40

40 columns color screen mode.

C80 = CO80

80 columns color screen mode.

CO40 = 1

40 columns color screen mode.

CO80 = 3

80 columns color screen mode.

ConsoleMaxX = 1024

ConsoleMaxY = 1024

Cyan = 3

Cyan color attribute

DarkGray = 8

Dark gray color attribute

Flushing = false

Font8x8 = 256

Internal ROM font mode

Green = 2

Green color attribute

LightBlue = 9

Light Blue color attribute

LightCyan = 11

Light cyan color attribute

LightGray = 7

Light gray color attribute

LightGreen = 10

Light green color attribute

LightMagenta = 13

Light magenta color attribute

LightRed = 12

Light red color attribute

Magenta = 5

Magenta color attribute

Mono = 7

Monochrome screen mode (hercules screens)

Red = 4

Red color attribute

ScreenHeight : LongInt = 25

Current screen height.

ScreenWidth : LongInt = 80

Current screen width

White = 15

White color attribute

Yellow = 14

Yellow color attribute

Types

```
PConsoleBuf = ^TConsoleBuf
```

```
TCharAttr = packed record
  ch : Char;
  attr : Byte;
end
```

```
TConsoleBuf = Array[0..ConsoleMaxX*ConsoleMaxY-1] of TCharAttr
```

Variables

```
CheckBreak : Boolean
```

Check for CTRL-Break keystroke. Not used.

```
CheckEOF : Boolean
```

Check for EOF on standard input. Not used.

```
CheckSnow : Boolean
```

Check snow on CGA screens. Not used.

```
ConsoleBuf : PConsoleBuf
```

```
DirectVideo : Boolean
```

The DirectVideo variable controls the writing to the screen. If it is `True`, the the cursor is set via direct port access. If `False`, then the BIOS is used. This is defined under dos only.

```
LastMode : Word = 3
```

The Lastmode variable tells you which mode was last selected for the screen. It is defined on DOS only.

```
TextAttr : Byte = $07
```

The TextAttr variable controls the attributes with which characters are written to screen.

```
WindMax : Word = $184f
```

Maximum window dimension

```
WindMaxX : DWord
```

Maximum window X size

```
WindMaxY : DWord
```

Maximum window Y size

WindMin : Word = \$0

Minimum window dimension

WindMinX : DWord

Minimum window X size

WindMinY : DWord

Minimum window Y size

3.3 Procedures and functions

AssignCrt

Synopsis: Assign file to CRT.

Declaration: `procedure AssignCrt(var F: Text)`

Visibility: default

Description: `AssignCrt` Assigns a file `F` to the console. Everything written to the file `F` goes to the console instead. If the console contains a window, everything is written to the window instead.

Errors: None.

See also: Window ([331](#))

Listing: `./crtex/ex1.pp`

```

Program Example1;
uses Crt;

{ Program to demonstrate the AssignCrt function. }

var
  F : Text;
begin
  AssignCrt(F);
  Rewrite(F); { Don't forget to open for output! }
  WriteLn(F, 'This is written to the Assigned File');
  Close(F);
end.

```

ClrEol

Synopsis: Clear from cursor position till end of line.

Declaration: `procedure ClrEol`

Visibility: default

Description: `ClrEol` clears the current line, starting from the cursor position, to the end of the window. The cursor doesn't move

Errors: None.

See also: `DelLine` ([324](#)), `InsLine` ([326](#)), `ClrScr` ([322](#))

Listing: `./crtex/ex9.pp`

```

Program Example9;
uses Crt;

{ Program to demonstrate the ClrEol function. }
var
  I,J : integer;

begin
  For I:=1 to 15 do
    For J:=1 to 80 do
      begin
        gotoxy(j,i);
        Write(j mod 10);
      end;
  Window(5,5,75,12);
  Write('This line will be cleared from ',
        ' here till the right of the window');
  GotoXY(27,WhereY);
  ReadKey;
  ClrEol;
  WriteLn;
end.
```

ClrScr

Synopsis: Clear current window.

Declaration: `procedure ClrScr`

Visibility: default

Description: `ClrScr` clears the current window (using the current colors), and sets the cursor in the top left corner of the current window.

Errors: None.

See also: `Window` ([331](#))

Listing: `./crtex/ex8.pp`

```

Program Example8;
uses Crt;

{ Program to demonstrate the ClrScr function. }

begin
  WriteLn('Press any key to clear the screen');
  ReadKey;
  ClrScr;
```

```
WriteIn('Have fun with the cleared screen');  
end.
```

cursorbig

Synopsis: Show big cursor

Declaration: `procedure cursorbig`

Visibility: default

Description: `CursorBig` makes the cursor a big rectangle. Not implemented on unixes.

Errors: None.

See also: `CursorOn` ([323](#)), `CursorOff` ([323](#))

cursoroff

Synopsis: Hide cursor

Declaration: `procedure cursoroff`

Visibility: default

Description: `CursorOff` switches the cursor off (i.e. the cursor is no longer visible). Not implemented on unixes.

Errors: None.

See also: `CursorOn` ([323](#)), `CursorBig` ([323](#))

cursoron

Synopsis: Display cursor

Declaration: `procedure cursoron`

Visibility: default

Description: `CursorOn` switches the cursor on. Not implemented on unixes.

Errors: None.

See also: `CursorBig` ([323](#)), `CursorOff` ([323](#))

Delay

Synopsis: Delay program execution.

Declaration: `procedure Delay(MS: Word)`

Visibility: default

Description: `Delay` waits a specified number of milliseconds. The number of specified seconds is an approximation, and may be off a lot, if system load is high.

Errors: None

See also: [Sound \(329\)](#), [NoSound \(327\)](#)

Listing: ./crtex/ex15.pp

```

Program Example15;
uses Crt;

{ Program to demonstrate the Delay function. }
var
    i : longint;
begin
    WriteLn( 'Counting Down' );
    for i:=10 downto 1 do
        begin
            WriteLn(i);
            Delay(1000); {Wait one second}
        end;
    WriteLn( 'BOOM!!! ' );
end.

```

DelLine

Synopsis: Delete line at cursor position.

Declaration: `procedure DelLine`

Visibility: default

Description: `DelLine` removes the current line. Lines following the current line are scrolled 1 line up, and an empty line is inserted at the bottom of the current window. The cursor doesn't move.

Errors: None.

See also: [ClrEol \(321\)](#), [InsLine \(326\)](#), [ClrScr \(322\)](#)

Listing: ./crtex/ex11.pp

```

Program Example10;
uses Crt;

{ Program to demonstrate the InsLine function. }

begin
    ClrScr;
    WriteLn;
    WriteLn( 'Line 1 ' );
    WriteLn( 'Line 2 ' );
    WriteLn( 'Line 2 ' );
    WriteLn( 'Line 3 ' );
    WriteLn;
    WriteLn( 'Oops, Line 2 is listed twice, ',
            ' let 's delete the line at the cursor position' );
    GotoXY(1,3);
    ReadKey;
    DelLine;
    GotoXY(1,10);
end.

```

GotoXY

Synopsis: Set cursor position on screen.

Declaration: `procedure GotoXY(X: Byte;Y: Byte)`

Visibility: default

Description: `GotoXY` positions the cursor at (X,Y), X in horizontal, Y in vertical direction relative to the origin of the current window. The origin is located at (1,1), the upper-left corner of the window.

Errors: None.

See also: [WhereX \(330\)](#), [WhereY \(331\)](#), [Window \(331\)](#)

Listing: `./crtex/ex6.pp`

```

Program Example6;
uses Crt;

{ Program to demonstrate the GotoXY function. }

begin
  ClrScr;
  GotoXY(10,10);
  Write('10,10');
  GotoXY(70,20);
  Write('70,20');
  GotoXY(1,22);
end.

```

HighVideo

Synopsis: Switch to highlighted text mode

Declaration: `procedure HighVideo`

Visibility: default

Description: `HighVideo` switches the output to highlighted text. (It sets the high intensity bit of the video attribute)

Errors: None.

See also: [TextColor \(330\)](#), [TextBackground \(329\)](#), [LowVideo \(327\)](#), [NormVideo \(327\)](#)

Listing: `./crtex/ex14.pp`

```

Program Example14;
uses Crt;

{ Program to demonstrate the LowVideo, HighVideo, NormVideo functions. }

begin
  LowVideo;
  WriteLn('This is written with LowVideo');
  HighVideo;
  WriteLn('This is written with HighVideo');
  NormVideo;
  WriteLn('This is written with NormVideo');
end.

```

InsLine

Synopsis: Insert an empty line at cursor position

Declaration: `procedure InsLine`

Visibility: `default`

Description: `InsLine` inserts an empty line at the current cursor position. Lines following the current line are scrolled 1 line down, causing the last line to disappear from the window. The cursor doesn't move.

Errors: None.

See also: `ClrEol` ([321](#)), `DelLine` ([324](#)), `ClrScr` ([322](#))

Listing: `./crtex/ex10.pp`

```

Program Example10;
uses Crt;

{ Program to demonstrate the InsLine function. }

begin
  ClrScr;
  WriteLn;
  WriteLn('Line 1 ');
  WriteLn('Line 3 ');
  WriteLn;
  WriteLn('Oops, forgot Line 2, let''s insert at the cursor postion');
  GotoXY(1,3);
  ReadKey;
  InsLine;
  Write('Line 2 ');
  GotoXY(1,10);
end.

```

KeyPressed

Synopsis: Check if there is a keypress in the keybuffer

Declaration: `function KeyPressed : Boolean`

Visibility: `default`

Description: `KeyPressed` scans the keyboard buffer and sees if a key has been pressed. If this is the case, `True` is returned. If not, `False` is returned. The `Shift`, `Alt`, `Ctrl` keys are not reported. The key is not removed from the buffer, and can hence still be read after the `KeyPressed` function has been called.

Errors: None.

See also: `ReadKey` ([328](#))

Listing: `./crtex/ex2.pp`

```

Program Example2;
uses Crt;

{ Program to demonstrate the KeyPressed function. }

```

```

begin
  WriteLn('Waiting until a key is pressed');
  repeat
    until KeyPressed;
  { The key is not Read,
    so it should also be outputted at the commandline}
end.

```

LowVideo

Synopsis: Switch to low intensity colors.

Declaration: `procedure LowVideo`

Visibility: default

Description: `LowVideo` switches the output to non-highlighted text. (It clears the high intensity bit of the video attribute)

For an example, see [HighVideo \(325\)](#)

Errors: None.

See also: [TextColor \(330\)](#), [TextBackground \(329\)](#), [HighVideo \(325\)](#), [NormVideo \(327\)](#)

NormVideo

Synopsis: Return to normal (startup) modus

Declaration: `procedure NormVideo`

Visibility: default

Description: `NormVideo` switches the output to the defaults, read at startup. (The defaults are read from the cursor position at startup)

For an example, see [HighVideo \(325\)](#)

Errors: None.

See also: [TextColor \(330\)](#), [TextBackground \(329\)](#), [LowVideo \(327\)](#), [HighVideo \(325\)](#)

NoSound

Synopsis: Stop system speaker

Declaration: `procedure NoSound`

Visibility: default

Description: `NoSound` stops the speaker sound. This call is not supported on all operating systems.

Errors: None.

See also: [Sound \(329\)](#)

Listing: `./crtex/ex16.pp`

```

Program Example16;
uses Crt;

{ Program to demonstrate the Sound and NoSound function. }

var
    i : longint;
begin
    WriteLn('You will hear some tones from your speaker');
    while (i<15000) do
        begin
            inc(i,500);
            Sound(i);
            Delay(100);
        end;
    WriteLn('Quiet now!');
    NoSound; {Stop noise}
end.

```

ReadKey

Synopsis: Read key from keybuffer

Declaration: `function ReadKey : Char`

Visibility: default

Description: ReadKey reads 1 key from the keyboard buffer, and returns this. If an extended or function key has been pressed, then the zero ASCII code is returned. You can then read the scan code of the key with a second ReadKey call.

Key mappings under Linux can cause the wrong key to be reported by ReadKey, so caution is needed when using ReadKey.

Errors: None.

See also: KeyPressed ([326](#))

Listing: ./crtex/ex3.pp

```

Program Example3;
uses Crt;

{ Program to demonstrate the ReadKey function. }

var
    ch : char;
begin
    writeln('Press Left/Right , Esc=Quit');
    repeat
        ch:=ReadKey;
        case ch of
            #0 : begin
                ch:=ReadKey; {Read ScanCode}
                case ch of
                    #75 : WriteLn('Left');
                    #77 : WriteLn('Right');
            end;
    until ch = #27;
end.

```

```

        end;
    end;
    #27 : WriteLn( 'ESC' );
    end;
    until ch=#27 {Esc}
end.

```

Sound

Synopsis: Sound system speaker

Declaration: `procedure Sound(Hz: Word)`

Visibility: default

Description: Sound sounds the speaker at a frequency of `hz`. Under Windows, a system sound is played and the frequency parameter is ignored. On other operating systems, this routine may not be implemented.

Errors: None.

See also: NoSound ([327](#))

TextBackground

Synopsis: Set text background

Declaration: `procedure TextBackground(Color: Byte)`

Visibility: default

Description: TextBackground sets the background color to `CL`. `CL` can be one of the predefined color constants.

Errors: None.

See also: TextColor ([330](#)), HighVideo ([325](#)), LowVideo ([327](#)), NormVideo ([327](#))

Listing: `./crtex/ex13.pp`

```

Program Example13;
uses Crt;

{ Program to demonstrate the TextBackground function. }

begin
    TextColor(White);
    WriteLn('This is written in with the default background color');
    TextBackground(Green);
    WriteLn('This is written in with a Green background');
    TextBackground(Brown);
    WriteLn('This is written in with a Brown background');
    TextBackground(Black);
    WriteLn('Back with a black background');
end.

```

TextColor

Synopsis: Set text color

Declaration: `procedure TextColor(Color: Byte)`

Visibility: default

Description: `TextColor` sets the foreground color to CL. CL can be one of the predefined color constants.

Errors: None.

See also: `TextBackground` ([329](#)), `HighVideo` ([325](#)), `LowVideo` ([327](#)), `NormVideo` ([327](#))

Listing: `./crtex/ex12.pp`

```
Program Example12;
uses Crt;

{ Program to demonstrate the TextColor function. }

begin
  WriteLn('This is written in the default color');
  TextColor(Red);
  WriteLn('This is written in Red');
  TextColor(White);
  WriteLn('This is written in White');
  TextColor(LightBlue);
  WriteLn('This is written in Light Blue');
end.
```

TextMode

Synopsis: Set screen mode.

Declaration: `procedure TextMode(Mode: Integer)`

Visibility: default

Description: `TextMode` sets the textmode of the screen (i.e. the number of lines and columns of the screen). The lower byte is use to set the VGA text mode.

This procedure is only implemented on dos.

Errors: None.

See also: `Window` ([331](#))

WhereX

Synopsis: Return X (horizontal) cursor position

Declaration: `function WhereX : Byte`

Visibility: default

Description: `WhereX` returns the current X-coordinate of the cursor, relative to the current window. The origin is (1,1), in the upper-left corner of the window.

Errors: None.

See also: [GotoXY \(325\)](#), [WhereY \(331\)](#), [Window \(331\)](#)

Listing: ./crtex/ex7.pp

```

Program Example7;
uses Crt;

{ Program to demonstrate the WhereX and WhereY functions. }

begin
  WriteLn( 'Cursor position: X= ',WhereX, ' Y= ',WhereY);
end.

```

WhereY

Synopsis: Return Y (vertical) cursor position

Declaration: `function WhereY : Byte`

Visibility: default

Description: WhereY returns the current Y-coordinate of the cursor, relative to the current window. The origin is (1,1), in the upper-left corner of the window.

Errors: None.

See also: [GotoXY \(325\)](#), [WhereX \(330\)](#), [Window \(331\)](#)

Listing: ./crtex/ex7.pp

```

Program Example7;
uses Crt;

{ Program to demonstrate the WhereX and WhereY functions. }

begin
  WriteLn( 'Cursor position: X= ',WhereX, ' Y= ',WhereY);
end.

```

Window

Synopsis: Create new window on screen.

Declaration: `procedure Window(X1: Byte;Y1: Byte;X2: Byte;Y2: Byte)`

Visibility: default

Description: Window creates a window on the screen, to which output will be sent. (X1, Y1) are the coordinates of the upper left corner of the window, (X2, Y2) are the coordinates of the bottom right corner of the window. These coordinates are relative to the entire screen, with the top left corner equal to (1,1). Further coordinate operations, except for the next Window call, are relative to the window's top left corner.

Errors: None.

See also: [GotoXY \(325\)](#), [WhereX \(330\)](#), [WhereY \(331\)](#), [ClrScr \(322\)](#)

Listing: ./crtex/ex5.pp

```
Program Example5;
uses Crt;

{ Program to demonstrate the Window function. }

begin
  ClrScr;
  WriteLn( 'Creating a window from 30,10 to 50,20' );
  Window(30,10,50,20);
  WriteLn( 'We are now writing in this small window we just created, we '+
    'can''t get outside it when writing long lines like this one' );
  Write( 'Press any key to clear the window' );
  ReadKey;
  ClrScr;
  Write( 'The window is cleared, press any key to restore to fullscreen' );
  ReadKey;
  { Full Screen is 80x25 }
  Window(1,1,80,25);
  Clrscr;
  WriteLn( 'Back in Full Screen' );
end.
```

Chapter 4

Reference for unit 'dateutils'

4.1 Used units

Table 4.1: Used units by unit 'dateutils'

Name	Page
math	601
sysutils	1071
Types	333

4.2 Overview

`DateUtils` contains a large number of date/time manipulation routines, all based on the `TDateTime` type. There are routines for date/time math, for comparing dates and times, for composing dates and decomposing dates in their constituent parts.

4.3 Constants, types and variables

Constants

`ApproxDaysPerMonth` : `Double` = 30.4375

Average number of days in a month, measured over a year. Used in `MonthsBetween` ([378](#)).

`ApproxDaysPerYear` : `Double` = 365.25

Average number of days in a year, measured over 4 years. Used in `YearsBetween` ([418](#)).

`DayFriday` = 5

ISO day number for Friday

`DayMonday` = 1

ISO day number for Monday

DaySaturday = 6

ISO day number for Saturday

DaysPerWeek = 7

Number of days in a week.

DaysPerYear : Array[Boolean] of Word = (365,366)

Array with number of days in a year. The boolean index indicates whether it is a leap year or not.

DaySunday = 7

ISO day number for Sunday

DayThursday = 4

ISO day number for Thursday

DayTuesday = 2

ISO day number for Tuesday

DayWednesday = 3

ISO day number for Wednesday

MonthsPerYear = 12

Number of months in a year

OneHour = 1 / HoursPerDay

One hour as a fraction of a day (suitable for TDateTime)

OneMillisecond = 1 / MSecsPerDay

One millisecond as a fraction of a day (suitable for TDateTime)

OneMinute = 1 / MinsPerDay

One minute as a fraction of a day (suitable for TDateTime)

OneSecond = 1 / SecsPerDay

One second as a fraction of a day (suitable for TDateTime)

RecodeLeaveFieldAsIs = High (Word)

Bitmask deciding what to do with each TDateTime field in recode routines

WeeksPerFortnight = 2

Number of weeks in fortnight

YearsPerCentury = 100

Number of years in a century

YearsPerDecade = 10

Number of years in a decade

YearsPerMillennium = 1000

Number of years in a millenium

4.4 Procedures and functions

CompareDate

Synopsis: Compare 2 dates, disregarding the time of day

Declaration: `function CompareDate(const A: TDateTime;const B: TDateTime)
: TValueRelationship`

Visibility: default

Description: CompareDate compares the date parts of two timestamps A and B and returns the following results:

< 0 if the day part of A is earlier than the day part of B.

0 if A and B are the on same day (times may differ) .

> 0 if the day part of A is later than the day part of B.

See also: CompareTime (337), CompareDateTime (336), SameDate (387), SameTime (389), SameDateTime (388)

Listing: ./datutex/ex99.pp

Program Example99;

{ This program demonstrates the CompareDate function }

Uses SysUtils, DateUtils;

Const

Fmt = 'dddd dd mmm yyyy';

Procedure Test(D1,D2 : TDateTime);

Var

Cmp : Integer;

begin

Write(FormatDateTime(Fmt,D1), ' is ');

Cmp:=CompareDate(D1,D2);

If Cmp<0 **then**

write('earlier than ')

```

    else if Cmp>0 then
        Write('later than ')
    else
        Write('equal to ');
    WriteLn(FormatDateTime(Fmt,D2));
end;

Var
    D,N : TDateTime;

Begin
    D:=Today;
    N:=Now;
    Test(D,D);
    Test(N,N);
    Test(D+1,D);
    Test(D-1,D);
    Test(D+OneSecond,D);
    Test(D-OneSecond,D);
    Test(N+OneSecond,N);
    Test(N-OneSecond,N);
End.

```

CompareDateTime

Synopsis: Compare 2 dates, taking into account the time of day

Declaration: `function CompareDateTime(const A: TDateTime;const B: TDateTime) : TValueRelationship`

Visibility: default

Description: CompareDateTime compares two timestamps A and B and returns the following results:

< 0 if A is earlier in date/time than B.
 0 if A and B are the same date/time .
 > 0 if A is later in date/time than B.

See also: CompareTime ([337](#)), CompareDate ([335](#)), SameDate ([387](#)), SameTime ([389](#)), SameDateTime ([388](#))

Listing: ./datutex/ex98.pp

Program Example98;

{ This program demonstrates the CompareDateTime function }

Uses SysUtils , DateUtils ;

Const

Fmt = 'dddd dd mmm yyyy hh:nn:ss.zzz' ;

Procedure Test(D1,D2 : TDateTime);

Var

Cmp : Integer;

```

begin
  Write(FormatDateTime(Fmt,D1), ' is ');
  Cmp:=CompareDateTime(D1,D2);
  If Cmp<0 then
    write('earlier than ')
  else if Cmp>0 then
    Write('later than ')
  else
    Write('equal to ');
  WriteIn(FormatDateTime(Fmt,D2));
end;

Var
  D,N : TDateTime;

Begin
  D:=Today;
  N:=Now;
  Test(D,D);
  Test(N,N);
  Test(D+1,D);
  Test(D-1,D);
  Test(D+OneSecond,D);
  Test(D-OneSecond,D);
  Test(N+OneSecond,N);
  Test(N-OneSecond,N);
End.

```

CompareTime

Synopsis: Compares two times of the day, disregarding the date part.

Declaration: `function CompareTime(const A: TDateTime;const B: TDateTime)
: TValueRelationship`

Visibility: default

Description: `CompareTime` compares the time parts of two timestamps A and B and returns the following results:

- < 0 if the time part of A is earlier than the time part of B.
- 0 if A and B have the same time part (dates may differ) .
- > 0 if the time part of A is later than the time part of B.

See also: `CompareDateTime` (336), `CompareDate` (335), `SameDate` (387), `SameTime` (389), `SameDateTime` (388)

Listing: ./datutex/ex100.pp

Program Example100;

{ This program demonstrates the CompareTime function }

Uses SysUtils , DateUtils ;

Const

```

    Fmt = 'dddd dd mmmm yyyy hh:nn:ss.zzz';

Procedure Test(D1,D2 : TDateTime);

Var
    Cmp : Integer;

begin
    Write(FormatDateTime(Fmt,D1), ' has ');
    Cmp:=CompareDateTime(D1,D2);
    If Cmp<0 then
        write('earlier time than ')
    else if Cmp>0 then
        Write('later time than ')
    else
        Write('equal time with ');
    WriteIn(FormatDateTime(Fmt,D2));
end;

Var
    D,N : TDateTime;

Begin
    D:=Today;
    N:=Now;
    Test(D,D);
    Test(N,N);
    Test(N+1,N);
    Test(N-1,N);
    Test(N+OneSecond,N);
    Test(N-OneSecond,N);
End.

```

DateOf

Synopsis: Extract the date part from a DateTime indication.

Declaration: `function DateOf(const AValue: TDateTime) : TDateTime`

Visibility: default

Description: DateOf extracts the date part from AValue and returns the result.

Since the TDateTime is actually a double with the date part encoded in the integer part, this operation corresponds to a call to Trunc.

See also: TimeOf ([398](#)), YearOf ([417](#)), MonthOf ([377](#)), DayOf ([339](#)), HourOf ([354](#)), MinuteOf ([373](#)), SecondOf ([390](#)), MilliSecondOf ([369](#))

Listing: ./datutex/ex1.pp

Program Example1;

{ This program demonstrates the DateOf function }

Uses SysUtils, DateUtils;

Begin

```
WriteIn( 'Date is : ', DateTimeToStr(DateOf(Now)));  
End.
```

DateTimeToJulianDate

Synopsis: Converts a TDateTime value to a Julian date representation

Declaration: `function DateTimeToJulianDate(const AValue: TDateTime) : Double`

Visibility: default

Description: Not yet implemented.

Errors: Currently, trying to use this function will raise an exception.

See also: [JulianDateToDateTime \(369\)](#), [TryJulianDateToDateTime \(403\)](#), [DateTimeToModifiedJulianDate \(339\)](#), [TryModifiedJulianDateToDateTime \(403\)](#)

DateTimeToModifiedJulianDate

Synopsis: Convert a TDateTime value to a modified Julian date representation

Declaration: `function DateTimeToModifiedJulianDate(const AValue: TDateTime) : Double`

Visibility: default

Description: Not yet implemented.

Errors: Currently, trying to use this function will raise an exception.

See also: [DateTimeToJulianDate \(339\)](#), [JulianDateToDateTime \(369\)](#), [TryJulianDateToDateTime \(403\)](#), [TryModifiedJulianDateToDateTime \(403\)](#)

DateTimeToUnix

Synopsis: Convert a TDateTime value to Unix epoch time

Declaration: `function DateTimeToUnix(const AValue: TDateTime) : Int64`

Visibility: default

Description: Not yet implemented.

Errors: Currently, trying to use this function will raise an exception.

See also: [UnixToDateTime \(405\)](#)

DayOf

Synopsis: Extract the day (of month) part from a DateTime value

Declaration: `function DayOf(const AValue: TDateTime) : Word`

Visibility: default

Description: DayOf returns the day of the month part of the AValue date/time indication. It is a number between 1 and 31.

For an example, see [YearOf \(417\)](#)

See also: YearOf ([417](#)), WeekOf ([405](#)), MonthOf ([377](#)), HourOf ([354](#)), MinuteOf ([373](#)), SecondOf ([390](#)), MilliSecondOf ([369](#))

DayOfTheMonth

Synopsis: Extract the day (of month) part of a DateTime value

Declaration: `function DayOfTheMonth(const AValue: TDateTime) : Word`

Visibility: default

Description: DayOfTheMonth returns the number of days that have passed since the start of the month till the moment indicated by AValue. This is a one-based number, i.e. the first day of the month will return 1.

For an example, see the WeekOfTheMonth ([405](#)) function.

See also: DayOfTheYear ([341](#)), WeekOfTheMonth ([405](#)), HourOfTheMonth ([355](#)), MinuteOfTheMonth ([374](#)), SecondOfTheMonth ([391](#)), MilliSecondOfTheMonth ([370](#))

DayOfTheWeek

Synopsis: Extracts the day of the week from a DateTime value

Declaration: `function DayOfTheWeek(const AValue: TDateTime) : Word`

Visibility: default

Description: DayOfTheWeek returns the number of days that have passed since the start of the week till the moment indicated by AValue. This is a one-based number, i.e. the first day of the week will return 1.

See also: DayOfTheYear ([341](#)), DayOfTheMonth ([340](#)), HourOfTheWeek ([355](#)), MinuteOfTheWeek ([375](#)), SecondOfTheWeek ([391](#)), MilliSecondOfTheWeek ([371](#))

Listing: ./datutex/ex42.pp

Program Example42;

{ This program demonstrates the WeekOfTheMonth function }

Uses SysUtils, DateUtils;

Var

N : TDateTime;

Begin

N:=Now;

WriteLn ('Day of the Week : ', DayOfTheWeek(N));

WriteLn ('Hour of the Week : ', HourOfTheWeek(N));

WriteLn ('Minute of the Week : ', MinuteOfTheWeek(N));

WriteLn ('Second of the Week : ', SecondOfTheWeek(N));

WriteLn ('MilliSecond of the Week : ',
MilliSecondOfTheWeek(N));

End.

DayOfTheYear

Synopsis: Extracts the day of the year from a TDateTime value

Declaration: `function DayOfTheYear(const AValue: TDateTime) : Word`

Visibility: default

Description: `DayOfTheYear` returns the number of days that have passed since the start of the year till the moment indicated by `AValue`. This is a one-based number, i.e. January 1 will return 1.

For an example, see the `WeekOfTheYear` (406) function.

See also: `WeekOfTheYear` (406), `HourOfTheYear` (356), `MinuteOfTheYear` (375), `SecondOfTheYear` (392), `MilliSecondOfTheYear` (371)

DaysBetween

Synopsis: Number of whole days between two DateTime values.

Declaration: `function DaysBetween(const ANow: TDateTime;const AThen: TDateTime) : Integer`

Visibility: default

Description: `DaysBetween` returns the number of whole days between `ANow` and `AThen`. This means the fractional part of a day (hours, minutes, etc.) is dropped.

See also: `YearsBetween` (418), `MonthsBetween` (378), `WeeksBetween` (407), `HoursBetween` (356), `MinutesBetween` (375), `SecondsBetween` (392), `MillisecondsBetween` (372)

Listing: `./datutex/ex58.pp`

Program `Example58;`

{ This program demonstrates the DaysBetween function }

Uses `SysUtils, DateUtils;`

Procedure `Test(ANow, AThen : TDateTime);`

begin

`Write('Number of days between ');`
 `Write(DateTimeToStr(AThen), ' and ', DateTimeToStr(ANow));`
 `WriteLn(' : ', DaysBetween(ANow, AThen));`
end;

Var

`D1, D2 : TDateTime;`

Begin

`D1:=Now;`
 `D2:=Today-23/24;`
 `Test(D1,D2);`
 `D2:=Today-1;`
 `Test(D1,D2);`
 `D2:=Today-25/24;`
 `Test(D1,D2);`
 `D2:=Today-26/24;`
 `Test(D1,D2);`

```
D2:=Today-5.4;
Test(D1,D2);
D2:=Today-2.5;
Test(D1,D2);
End.
```

DaysInAMonth

Synopsis: Number of days in a month of a certain year.

Declaration: `function DaysInAMonth(const AYear: Word;const AMonth: Word) : Word`

Visibility: default

Description: `DaysInAMonth` returns the number of days in the month `AMonth` in the year `AYear`. The return value takes leap years into account.

See also: `WeeksInAYear` ([407](#)), `WeeksInYear` ([408](#)), `DaysInYear` ([343](#)), `DaysInAYear` ([342](#)), `DaysInMonth` ([343](#))

Listing: `./datutex/ex17.pp`

Program `Example17;`

```
{ This program demonstrates the DaysInAMonth function }
```

```
Uses SysUtils, DateUtils;
```

```
Var
```

```
Y,M : Word;
```

```
Begin
```

```
For Y:=1992 to 2010 do
```

```
For M:=1 to 12 do
```

```
WriteLn(LongMonthNames[m], ' ', Y, ' has ', DaysInAMonth(Y,M), ' days. ');
```

```
End.
```

DaysInAYear

Synopsis: Number of days in a particular year.

Declaration: `function DaysInAYear(const AYear: Word) : Word`

Visibility: default

Description: `DaysInAYear` returns the number of days in the year `AYear`. The return value is either 365 or 366.

See also: `WeeksInAYear` ([407](#)), `WeeksInYear` ([408](#)), `DaysInYear` ([343](#)), `DaysInMonth` ([343](#)), `DaysInAMonth` ([342](#))

Listing: `./datutex/ex15.pp`

Program `Example15;`

```
{ This program demonstrates the DaysInAYear function }
```

Uses SysUtils , DateUtils ;

Var

Y : Word;

Begin

For Y:=1992 to 2010 do

WriteLn(Y, ' has ', DaysInAYear(Y), ' days. ');

End.

DaysInMonth

Synopsis: Return the number of days in the month in which a date occurs.

Declaration: function DaysInMonth(const AValue: TDateTime) : Word

Visibility: default

Description: DaysInMonth returns the number of days in the month in which AValue falls. The return value takes leap years into account.

See also: WeeksInAYear ([407](#)), WeeksInYear ([408](#)), DaysInYear ([343](#)), DaysInAYear ([342](#)), DaysInAMonth ([342](#))

Listing: ./datutex/ex16.pp

Program Example16;

{ This program demonstrates the DaysInMonth function }

Uses SysUtils , DateUtils ;

Var

Y,M : Word;

Begin

For Y:=1992 to 2010 do

For M:=1 to 12 do

WriteLn(LongMonthNames[m], ' ', Y, ' has ', DaysInMonth(EncodeDate(Y,M,1)), ' days. ');

End.

DaysInYear

Synopsis: Return the number of days in the year in which a date occurs.

Declaration: function DaysInYear(const AValue: TDateTime) : Word

Visibility: default

Description: daysInYear returns the number of days in the year part of AValue. The return value is either 365 or 366.

See also: WeeksInAYear ([407](#)), WeeksInYear ([408](#)), DaysInAYear ([342](#)), DaysInMonth ([343](#)), DaysInAMonth ([342](#))

Listing: ./datutex/ex14.pp

Program Example14;

{ This program demonstrates the DaysInYear function }

Uses SysUtils, DateUtils;

Var

Y : Word;

Begin

For Y:=1992 to 2010 do

WriteLn(Y, ' has ', DaysInYear(EncodeDate(Y,1,1)), ' days. ');

End.

DaySpan

Synopsis: Calculate the approximate number of days between two DateTime values.

Declaration: function DaySpan(const ANow: TDateTime;const AThen: TDateTime) : Double

Visibility: default

Description: DaySpan returns the number of Days between ANow and AThen, including any fractional parts of a Day.

See also: YearSpan ([419](#)), MonthSpan ([379](#)), WeekSpan ([409](#)), HourSpan ([357](#)), MinuteSpan ([376](#)), SecondSpan ([393](#)), MilliSecondSpan ([372](#)), DaysBetween ([341](#))

Listing: ./datutex/ex66.pp

Program Example66;

{ This program demonstrates the DaySpan function }

Uses SysUtils, DateUtils;

Procedure Test(ANow, AThen : TDateTime);

begin

Write('Number of days between ');

Write(DateTimeToStr(AThen), ' and ', DateTimeToStr(ANow));

WriteLn(' : ', DaySpan(ANow, AThen));

end;

Var

D1, D2 : TDateTime;

Begin

D1:=Now;

D2:=Today - 23/24;

Test(D1, D2);

D2:=Today - 1;

Test(D1, D2);

D2:=Today - 25/24;

Test(D1, D2);

D2:=Today - 26/24;

```

Test(D1,D2);
D2:=Today-5.4;
Test(D1,D2);
D2:=Today-2.5;
Test(D1,D2);
End.

```

DecodeDateDay

Synopsis: Decode a DateTime value in year and year of day.

Declaration: `procedure DecodeDateDay(const AValue: TDateTime; var AYear: Word; var ADayOfYear: Word)`

Visibility: default

Description: DecodeDateDay decomposes the date indication in AValue and returns the various components in AYear, ADayOfYear.

See also: EncodeDateTime (348), EncodeDateMonthWeek (348), EncodeDateWeek (349), EncodeDateDay (348), DecodeDateTime (346), DecodeDateWeek (346), DecodeDateMonthWeek (345)

Listing: ./datutex/ex83.pp

Program Example83;

{ This program demonstrates the DecodeDateDay function }

Uses SysUtils, DateUtils;

Var

Y, DoY : Word;
TS : TDateTime;

Begin

DecodeDateDay(**Now**, Y, DoY);
TS:=EncodeDateDay(Y, DoY);
WriteLn('Today is : ', **DateToStr**(TS));

End.

DecodeDateMonthWeek

Synopsis: Decode a DateTime value in a month, week of month and day of week

Declaration: `procedure DecodeDateMonthWeek(const AValue: TDateTime; var AYear: Word; var AMonth: Word; var AWeekOfMonth: Word; var ADayOfWeek: Word)`

Visibility: default

Description: DecodeDateMonthWeek decomposes the date indication in AValue and returns the various components in AYear, AMonth, AWeekOfMonth and ADayOfWeek.

See also: EncodeDateTime (348), EncodeDateMonthWeek (348), EncodeDateWeek (349), EncodeDateDay (348), DecodeDateTime (346), DecodeDateWeek (346), DecodeDateDay (345)

Listing: ./datutex/ex85.pp

Program Example85;

{ This program demonstrates the DecodeDateMonthWeek function }

Uses SysUtils, DateUtils;

Var

Y,M,WoM,Dow : Word;
TS : TDateTime;

Begin

DecodeDateMonthWeek(**Now**, Y, M, WoM, DoW);
TS:=EncodeDateMonthWeek(Y, M, WoM, Dow);
WriteIn('Today is : ', **DateToStr**(TS));

End.

DecodeDateTime

Synopsis: Decode a datetime value in a date and time value

Declaration: procedure DecodeDateTime(const AValue: TDateTime; var AYear: Word;
var AMonth: Word; var ADay: Word; var AHour: Word;
var AMinute: Word; var ASecond: Word;
var AMilliSecond: Word)

Visibility: default

Description: DecodeDateTime decomposes the date/time indication in AValue and returns the various components in AYear, AMonth, ADay, AHour, AMinute, ASecond, AMilliSecond

See also: EncodeDateTime (348), EncodeDateMonthWeek (348), EncodeDateWeek (349), EncodeDateDay (348), DecodeDateWeek (346), DecodeDateDay (345), DecodeDateMonthWeek (345)

Listing: ./datutex/ex79.pp

Program Example79;

{ This program demonstrates the DecodeDateTime function }

Uses SysUtils, DateUtils;

Var

Y,Mo,D,H,Mi,S,MS : Word;
TS : TDateTime;

Begin

DecodeDateTime(**Now**, Y, Mo, D, H, Mi, S, MS);
TS:=EncodeDateTime(Y, Mo, D, H, Mi, S, MS);
WriteIn('Now is : ', **DateTimeToStr**(TS));

End.

DecodeDateWeek

Synopsis: Decode a DateTime value in a week of year and day of week.

Declaration: `procedure DecodeDateWeek(const AValue: TDateTime; var AYear: Word;
var AWeekOfYear: Word; var ADayOfWeek: Word)`

Visibility: default

Description: DecodeDateWeek decomposes the date indication in AValue and returns the various components in AYear, AWeekOfYear, ADayOfWeek.

See also: EncodeDateTime (348), EncodeDateMonthWeek (348), EncodeDateWeek (349), EncodeDateDay (348), DecodeDateTime (346), DecodeDateDay (345), DecodeDateMonthWeek (345)

Listing: ./datutex/ex81.pp

Program Example81;

{ This program demonstrates the DecodeDateWeek function }

Uses SysUtils, DateUtils;

Var

Y, W, Dow : Word;
TS : TDateTime;

Begin

DecodeDateWeek(**Now**, Y, W, Dow);
TS := EncodeDateWeek(Y, W, Dow);
WriteLn('Today is : ', **DateToStr**(TS));

End.

DecodeDayOfWeekInMonth

Synopsis: Decode a DateTime value in year, month, day of week parts

Declaration: `procedure DecodeDayOfWeekInMonth(const AValue: TDateTime;
var AYear: Word; var AMonth: Word;
var ANthDayOfWeek: Word;
var ADayOfWeek: Word)`

Visibility: default

Description: DecodeDayOfWeekInMonth decodes the date AValue in a AYear, AMonth, ADayOfWeek and ANthDayOfWeek. (This is the N-th time that this weekday occurs in the month, e.g. the third saturday of the month.)

See also: NthDayOfWeek (379), EncodeDateMonthWeek (348), #rtl.sysutils.DayOfWeek (1114), EncodeDayOfWeekInMonth (349), TryEncodeDayOfWeekInMonth (402)

Listing: ./datutex/ex105.pp

Program Example105;

{ This program demonstrates the DecodeDayOfWeekInMonth function }

Uses SysUtils, DateUtils;

Var

Y, M, NDoW, DoW : Word;

```

D : TDateTime;
Begin
  DecodeDayOfWeekInMonth (Date , Y, M, NDoW, DoW);
  D:= EncodeDayOfWeekInMonth (Y, M, NDoW, DoW);
  Write (DateToStr (D), ' is the ', NDow, '-th ');
  WriteLn (formatdateTime ( 'dddd', D), ' of the month. ');
End.

```

EncodeDateDay

Synopsis: Encodes a year and day of year to a DateTime value

Declaration: `function EncodeDateDay(const AYear: Word;const ADayOfYear: Word)
: TDateTime`

Visibility: default

Description: EncodeDateDay encodes the values AYear and ADayOfYear to a date value and returns this value.

For an example, see DecodeDateDay (345).

Errors: If any of the arguments is not valid, then an EConvertError exception is raised.

See also: EncodeDateMonthWeek (348), DecodeDateDay (345), EncodeDateTime (348), EncodeDateWeek (349), TryEncodeDateTime (401), TryEncodeDateMonthWeek (400), TryEncodeDateWeek (402)

EncodeDateMonthWeek

Synopsis: Encodes a year, month, week of month and day of week to a DateTime value

Declaration: `function EncodeDateMonthWeek(const AYear: Word;const AMonth: Word;
const AWeekOfMonth: Word;
const ADayOfWeek: Word) : TDateTime`

Visibility: default

Description: EncodeDateTime encodes the values AYearAMonth, WeekOfMonth,ADayOfWeek, to a date value and returns this value.

For an example, see DecodeDateMonthWeek (345).

Errors: If any of the arguments is not valid, then an EConvertError exception is raised.

See also: DecodeDateMonthWeek (345), EncodeDateTime (348), EncodeDateWeek (349), EncodeDateDay (348), TryEncodeDateTime (401), TryEncodeDateWeek (402), TryEncodeDateMonthWeek (400), TryEncodeDateDay (400), NthDayOfWeek (379)

EncodeDateTime

Synopsis: Encodes a DateTime value from all its parts

Declaration: `function EncodeDateTime(const AYear: Word;const AMonth: Word;
const ADay: Word;const AHour: Word;
const AMinute: Word;const ASecond: Word;
const AMilliSecond: Word) : TDateTime`

Visibility: default

Description: `EncodeDateTime` encodes the values `AYearAMonth`, `ADay`, `AHour`, `AMinute`, `ASecond` and `AMilliSecond` to a date/time value and returns this value.

For an example, see `DecodeDateTime` (346).

Errors: If any of the arguments is not valid, then an `EConvertError` exception is raised.

See also: `DecodeDateTime` (346), `EncodeDateMonthWeek` (348), `EncodeDateWeek` (349), `EncodeDateDay` (348), `TryEncodeDateTime` (401), `TryEncodeDateWeek` (402), `TryEncodeDateDay` (400), `TryEncodeDateMonthWeek` (400)

EncodeDateWeek

Synopsis: Encode a `TDateTime` value from a year, week and day of week triplet

Declaration:

```
function EncodeDateWeek(const AYear: Word; const AWeekOfYear: Word;
                        const ADayOfWeek: Word) : TDateTime
function EncodeDateWeek(const AYear: Word; const AWeekOfYear: Word)
                        : TDateTime
```

Visibility: default

Description: `EncodeDateWeek` encodes the values `AYear`, `AWeekOfYear` and `ADayOfWeek` to a date value and returns this value.

For an example, see `DecodeDateWeek` (346).

Errors: If any of the arguments is not valid, then an `EConvertError` exception is raised.

See also: `EncodeDateMonthWeek` (348), `DecodeDateWeek` (346), `EncodeDateTime` (348), `EncodeDateDay` (348), `TryEncodeDateTime` (401), `TryEncodeDateWeek` (402), `TryEncodeDateMonthWeek` (400)

EncodeDayOfWeekInMonth

Synopsis: Encodes a year, month, week, day of week specification to a `TDateTime` value

Declaration:

```
function EncodeDayOfWeekInMonth(const AYear: Word; const AMonth: Word;
                                const ANthDayOfWeek: Word;
                                const ADayOfWeek: Word) : TDateTime
```

Visibility: default

Description: `EncodeDayOfWeekInMonth` encodes `AYear`, `AMonth`, `ADayOfWeek` and `ANthDayOfWeek` to a valid date stamp and returns the result.

`ANthDayOfWeek` is the N-th time that this weekday occurs in the month, e.g. the third saturday of the month.

For an example, see `DecodeDayOfWeekInMonth` (347).

Errors: If any of the values is not in range, then an `EConvertError` exception will be raised.

See also: `NthDayOfWeek` (379), `EncodeDateMonthWeek` (348), `#rtl.sysutils.DayOfWeek` (1114), `DecodeDayOfWeekInMonth` (347), `TryEncodeDayOfWeekInMonth` (402)

EndOfDay

Synopsis: Calculates a DateTime value representing the end of a specified day

Declaration: `function EndOfDay(const AYear: Word;const AMonth: Word;
const ADay: Word) : TDateTime; Overload
function EndOfDay(const AYear: Word;const ADayOfYear: Word) : TDateTime
; Overload`

Visibility: default

Description: EndOfDay returns a TDateTime value with the date/time indication of the last moment (23:59:59.999) of the day given by AYear, AMonth, ADay.

The day may also be indicated with a AYear, ADayOfYear pair.

See also: StartOfDay (396), StartOfDay (394), StartOfTheWeek (397), StartOfAWeek (395), StartOfA-Month (395), StartOfTheMonth (397), EndOfTheWeek (353), EndOfAWeek (351), EndOfTheYear (354), EndOfAYear (352), EndOfTheMonth (353), EndOfAMonth (350), EndOfTheDay (352)

Listing: ./datutex/ex39.pp

Program Example39;

{ This program demonstrates the EndOfDay function }

Uses SysUtils , DateUtils ;

Const

Fmt = ' "End of the day : " dd mmm yyyy hh:nn:ss ';

Var

Y,M,D : Word;

Begin

Y:=YearOf(Today);

M:=MonthOf(Today);

D:=DayOf(Today);

WriteIn (FormatDateTime (Fmt, EndOfDay(Y,M,D)));

DecodeDateDay(Today, Y,D);

WriteIn (FormatDateTime (Fmt, EndOfDay(Y,D)));

End.

EndOfAMonth

Synopsis: Calculate a datetime value representing the last day of the indicated month

Declaration: `function EndOfAMonth(const AYear: Word;const AMonth: Word) : TDateTime`

Visibility: default

Description: EndOfAMonth returns a TDateTime value with the date of the last day of the month indicated by the AYear, AMonth pair.

See also: StartOfTheMonth (397), StartOfAMonth (395), EndOfTheMonth (353), EndOfTheYear (354), End-OfAYear (352), StartOfAWeek (395), StartOfTheWeek (397)

Listing: ./datutex/ex31.pp

Program Example31;

{ This program demonstrates the EndOfAMonth function }

Uses SysUtils, DateUtils;

Const

 Fmt = ' "Last day of this month : " dd mmmm yyyy ';

Var

 Y,M : Word;

Begin

 Y:=YearOf(Today);

 M:=MonthOf(Today);

WriteLn (FormatDateTime (Fmt, EndOfAMonth(Y,M)));

End.

EndOfAWeek

Synopsis: Return the last moment of day of the week, given a year and a week in the year.

Declaration: `function EndOfAWeek(const AYear: Word;const AWeekOfYear: Word;
 const ADayOfWeek: Word) : TDateTime
function EndOfAWeek(const AYear: Word;const AWeekOfYear: Word)
 : TDateTime`

Visibility: default

Description: EndOfAWeek returns a TDateTime value with the date of the last moment (23:59:59:999) on the indicated day of the week indicated by the AYear, AWeek, ADayOfWeek values.

The default value for ADayOfWeek is 7.

See also: StartOfTheWeek ([397](#)), EndOfTheWeek ([353](#)), EndOfAWeek ([351](#)), StartOfAMonth ([395](#)), EndOfTheYear ([354](#)), EndOfAYear ([352](#)), EndOfTheMonth ([353](#)), EndOfAMonth ([350](#))

Listing: ./datutex/ex35.pp

Program Example35;

{ This program demonstrates the EndOfAWeek function }

Uses SysUtils, DateUtils;

Const

 Fmt = ' "Last day of this week : " dd mmmm yyyy hh:nn:ss ';

 Fmt2 = ' "Last-1 day of this week : " dd mmmm yyyy hh:nn:ss ';

Var

 Y,W : Word;

Begin

 Y:=YearOf(Today);

 W:=WeekOf(Today);

WriteLn (FormatDateTime (Fmt, EndOfAWeek(Y,W)));

WriteLn (FormatDateTime (Fmt2, EndOfAWeek(Y,W,6)));

End.

EndOfYear

Synopsis: Calculate a DateTime value representing the last day of a year

Declaration: `function EndOfYear(const AYear: Word) : TDateTime`

Visibility: default

Description: `StartOfYear` returns a `TDateTime` value with the date of the last day of the year `AYear` (December 31).

See also: `StartOfYear` ([398](#)), `EndOfYear` ([354](#)), `EndOfYear` ([352](#)), `EndOfMonth` ([353](#)), `EndOfMonth` ([350](#)), `StartOfWeek` ([395](#)), `StartOfWeek` ([397](#))

Listing: `./datutex/ex27.pp`

Program `Example27;`

{ This program demonstrates the EndOfYear function }

Uses `SysUtils, DateUtils;`

Const

`Fmt = 'Last day of this year : "dd mmm yyyy';`

Begin

`WriteIn (FormatDateTime (Fmt, EndOfYear (YearOf (Today))));`

End.

EndOfDay

Synopsis: Calculate a datetime value that represents the end of a given day.

Declaration: `function EndOfDay(const AValue: TDateTime) : TDateTime`

Visibility: default

Description: `EndOfDay` extracts the date part of `AValue` and returns a `TDateTime` value with the date/-time indication of the last moment (23:59:59.999) of this day.

See also: `StartOfDay` ([396](#)), `StartOfDay` ([394](#)), `StartOfWeek` ([397](#)), `StartOfWeek` ([395](#)), `StartOfMonth` ([395](#)), `StartOfMonth` ([397](#)), `EndOfWeek` ([353](#)), `EndOfWeek` ([351](#)), `EndOfYear` ([354](#)), `EndOfYear` ([352](#)), `EndOfMonth` ([353](#)), `EndOfMonth` ([350](#)), `EndOfDay` ([350](#))

Listing: `./datutex/ex37.pp`

Program `Example37;`

{ This program demonstrates the EndOfDay function }

Uses `SysUtils, DateUtils;`

Const

`Fmt = 'End of the day : "dd mmm yyyy hh:nn:ss';`

Begin

`WriteIn (FormatDateTime (Fmt, EndOfDay (Today)));`

End.

EndOfTheMonth

Synopsis: Calculate a `DateTime` value representing the last day of the month, given a day in that month.

Declaration: `function EndOfTheMonth(const AValue: TDateTime) : TDateTime`

Visibility: default

Description: `EndOfTheMonth` extracts the year and month parts of `AValue` and returns a `TDateTime` value with the date of the first day of that year and month as the `EndOfAMonth` (350) function.

See also: `StartOfAMonth` (395), `StartOfTheMonth` (397), `EndOfAMonth` (350), `EndOfTheYear` (354), `EndOfAYear` (352), `StartOfAWeek` (395), `StartOfTheWeek` (397)

Listing: `./datutex/ex29.pp`

Program `Example29;`

`{ This program demonstrates the EndOfTheMonth function }`

Uses `SysUtils, DateUtils;`

Const

`Fmt = ' "last day of this month : " dd mmm yyyy ';`

Begin

`WriteLn (FormatDateTime (Fmt, EndOfTheMonth (Today)));`

End.

EndOfTheWeek

Synopsis: Calculate a `DateTime` value which represents the end of a week, given a date in that week.

Declaration: `function EndOfTheWeek(const AValue: TDateTime) : TDateTime`

Visibility: default

Description: `EndOfTheWeek` extracts the year and week parts of `AValue` and returns a `TDateTime` value with the date of the last day of that week as the `EndOfAWeek` (351) function.

See also: `StartOfAWeek` (395), `StartOfTheWeek` (397), `EndOfAWeek` (351), `StartOfAMonth` (395), `EndOfTheYear` (354), `EndOfAYear` (352), `EndOfTheMonth` (353), `EndOfAMonth` (350)

Listing: `./datutex/ex33.pp`

Program `Example33;`

`{ This program demonstrates the EndOfTheWeek function }`

Uses `SysUtils, DateUtils;`

Const

`Fmt = ' "last day of this week : " dd mmm yyyy ';`

Begin

`WriteLn (FormatDateTime (Fmt, EndOfTheWeek (Today)));`

End.

EndOfTheYear

Synopsis: Calculate a `DateTime` value representing the last day of a year, given a date in that year.

Declaration: `function EndOfTheYear(const AValue: TDateTime) : TDateTime`

Visibility: default

Description: `EndOfTheYear` extracts the year part of `AValue` and returns a `TDateTime` value with the date of the last day of that year (December 31), as the `EndOfAYear` (352) function.

See also: `StartOfAYear` (396), `StartOfTheYear` (398), `EndOfTheMonth` (353), `EndOfAMonth` (350), `StartOfAWeek` (395), `StartOfTheWeek` (397), `EndOfAYear` (352)

Listing: `./datutex/ex25.pp`

Program `Example25;`

`{ This program demonstrates the EndOfTheYear function }`

Uses `SysUtils, DateUtils;`

Const

`Fmt = ' "Last day of this year : " dd mmmm yyyy ';`

Begin

`WriteIn (FormatDateTime (Fmt, EndOfTheYear (Today)));`

End.

HourOf

Synopsis: Extract the hour part from a `DateTime` value.

Declaration: `function HourOf(const AValue: TDateTime) : Word`

Visibility: default

Description: `HourOf` returns the hour of the day part of the `AValue` date/time indication. It is a number between 0 and 23.

For an example, see `YearOf` (417)

See also: `YearOf` (417), `WeekOf` (405), `MonthOf` (377), `DayOf` (339), `MinuteOf` (373), `SecondOf` (390), `MilliSecondOf` (369)

HourOfTheDay

Synopsis: Calculate the hour of a given `DateTime` value

Declaration: `function HourOfTheDay(const AValue: TDateTime) : Word`

Visibility: default

Description: `HourOfTheDay` returns the number of hours that have passed since the start of the day till the moment indicated by `AValue`. This is a zero-based number, i.e. 00:59:59 will return 0.

See also: `HourOfTheYear` (356), `HourOfTheMonth` (355), `HourOfTheWeek` (355), `MinuteOfTheDay` (373), `SecondOfTheDay` (390), `MilliSecondOfTheDay` (369)

Listing: ./datutex/ex43.pp

Program Example43;

{ This program demonstrates the HourOfDay function }

Uses SysUtils , DateUtils ;

Var

N : TDateTime;

Begin

N:=Now;

WriteLn('Hour of the Day : ', HourOfDay(N));

WriteLn('Minute of the Day : ', MinuteOfDay(N));

WriteLn('Second of the Day : ', SecondOfDay(N));

WriteLn('MilliSecond of the Day : ',
MilliSecondOfDay(N));

End.

HourOfTheMonth

Synopsis: Calculate the number of hours passed since the start of the month.

Declaration: function HourOfTheMonth(const AValue: TDateTime) : Word

Visibility: default

Description: HourOfTheMonth returns the number of hours that have passed since the start of the month till the moment indicated by AValue. This is a zero-based number, i.e. 00:59:59 on the first day of the month will return 0.

For an example, see the WeekOfTheMonth (405) function.

See also: WeekOfTheMonth (405), DayOfTheMonth (340), MinuteOfTheMonth (374), SecondOfTheMonth (391), MilliSecondOfTheMonth (370)

HourOfTheWeek

Synopsis: Calculate the number of hours elapsed since the start of the week.

Declaration: function HourOfTheWeek(const AValue: TDateTime) : Word

Visibility: default

Description: HourOfTheWeek returns the number of hours that have passed since the start of the Week till the moment indicated by AValue. This is a zero-based number, i.e. 00:59:59 on the first day of the week will return 0.

For an example, see the DayOfTheWeek (340) function.

See also: HourOfTheYear (356), HourOfTheMonth (355), HourOfDay (354), DayOfTheWeek (340), MinuteOfTheWeek (375), SecondOfTheWeek (391), MilliSecondOfTheWeek (371)

HourOfTheYear

Synopsis: Calculate the number of hours passed since the start of the year.

Declaration: `function HourOfTheYear(const AValue: TDateTime) : Word`

Visibility: default

Description: `HourOfTheYear` returns the number of hours that have passed since the start of the year (January 1, 00:00:00) till the moment indicated by `AValue`. This is a zero-based number, i.e. January 1 00:59:59 will return 0.

For an example, see the `WeekOfTheYear` (406) function.

See also: `WeekOfTheYear` (406), `DayOfTheYear` (341), `MinuteOfTheYear` (375), `SecondOfTheYear` (392), `MilliSecondOfTheYear` (371)

HoursBetween

Synopsis: Calculate the number of whole hours between two `DateTime` values.

Declaration: `function HoursBetween(const ANow: TDateTime;const AThen: TDateTime) : Int64`

Visibility: default

Description: `HoursBetween` returns the number of whole hours between `ANow` and `AThen`. This means the fractional part of an hour (minutes,seconds etc.) is dropped.

See also: `YearsBetween` (418), `MonthsBetween` (378), `WeeksBetween` (407), `DaysBetween` (341), `MinutesBetween` (375), `SecondsBetween` (392), `MillisecondsBetween` (372)

Listing: `./datutex/ex59.pp`

Program Example59;

{ This program demonstrates the HoursBetween function }

Uses SysUtils, DateUtils;

Procedure Test(ANow, AThen : TDateTime);

begin

 Write('Number of hours between ');

 Write(DateTimeToStr(AThen), ' and ', DateTimeToStr(ANow));

 WriteLn(' : ', HoursBetween(ANow, AThen));

end;

Var

 D1,D2 : TDateTime;

Begin

 D1:=Now;

 D2:=D1-(59*OneMinute);

 Test(D1,D2);

 D2:=D1-(61*OneMinute);

 Test(D1,D2);

 D2:=D1-(122*OneMinute);

 Test(D1,D2);

 D2:=D1-(306*OneMinute);

```

    Test(D1,D2);
    D2:=D1-(5.4*OneHour);
    Test(D1,D2);
    D2:=D1-(2.5*OneHour);
    Test(D1,D2);
End.

```

HourSpan

Synopsis: Calculate the approximate number of hours between two DateTime values.

Declaration: `function HourSpan(const ANow: TDateTime;const AThen: TDateTime) : Double`

Visibility: default

Description: HourSpan returns the number of Hours between ANow and AThen, including any fractional parts of a Hour.

See also: YearSpan ([419](#)), MonthSpan ([379](#)), WeekSpan ([409](#)), DaySpan ([344](#)), MinuteSpan ([376](#)), SecondSpan ([393](#)), MilliSecondSpan ([372](#)), HoursBetween ([356](#))

Listing: ./datutex/ex67.pp

Program Example67;

{ This program demonstrates the HourSpan function }

Uses SysUtils, DateUtils;

Procedure Test(ANow, AThen : TDateTime);

begin

```

    Write( 'Number of hours between ');
    Write( DateTimeToStr(AThen), ' and ', DateTimeToStr(ANow));
    Writeln( ' : ', HourSpan(ANow, AThen));
end;

```

Var

D1,D2 : TDateTime;

Begin

```

    D1:=Now;
    D2:=D1-(59*OneMinute);
    Test(D1,D2);
    D2:=D1-(61*OneMinute);
    Test(D1,D2);
    D2:=D1-(122*OneMinute);
    Test(D1,D2);
    D2:=D1-(306*OneMinute);
    Test(D1,D2);
    D2:=D1-(5.4*OneHour);
    Test(D1,D2);
    D2:=D1-(2.5*OneHour);
    Test(D1,D2);
End.

```

IncDay

Synopsis: Increase a DateTime value with a number of days.

Declaration: `function IncDay(const AValue: TDateTime;const ANumberOfDays: Integer)
: TDateTime
function IncDay(const AValue: TDateTime) : TDateTime`

Visibility: default

Description: `IncDay` adds `ANumberOfDays` days to `AValue` and returns the resulting date/time. `ANumberOfDays` can be positive or negative.

See also: `IncYear` (361), `#rtl.sysutils.IncMonth` (1154), `IncWeek` (360), `IncHour` (358), `IncMinute` (359), `IncSecond` (360), `IncMilliSecond` (359)

Listing: `./datutex/ex74.pp`

Program `Example74;`

`{ This program demonstrates the IncDay function }`

Uses `SysUtils, DateUtils;`

Begin

`WriteLn('One Day from today is ', DateTimeToStr(IncDay(Today, 1)));`

`WriteLn('One Day ago from today is ', DateTimeToStr(IncDay(Today, -1)));`

End.

IncHour

Synopsis: Increase a DateTime value with a number of hours.

Declaration: `function IncHour(const AValue: TDateTime;const ANumberOfHours: Int64)
: TDateTime
function IncHour(const AValue: TDateTime) : TDateTime`

Visibility: default

Description: `IncHour` adds `ANumberOfHours` hours to `AValue` and returns the resulting date/time. `ANumberOfHours` can be positive or negative.

See also: `IncYear` (361), `#rtl.sysutils.IncMonth` (1154), `IncWeek` (360), `IncDay` (358), `IncMinute` (359), `IncSecond` (360), `IncMilliSecond` (359)

Listing: `./datutex/ex75.pp`

Program `Example75`

`;`

`{ This program demonstrates the IncHour function }`

Uses `SysUtils, DateUtils;`

Begin

`WriteLn('One Hour from now is ', DateTimeToStr(IncHour(Now, 1)));`

`WriteLn('One Hour ago from now is ', DateTimeToStr(IncHour(Now, -1)));`

End.

IncMilliSecond

Synopsis: Increase a DateTime value with a number of milliseconds.

Declaration: `function IncMilliSecond(const AValue: TDateTime;
const ANumberOfMilliseconds: Int64) : TDateTime
function IncMilliSecond(const AValue: TDateTime) : TDateTime`

Visibility: default

Description: IncMilliSecond adds ANumberOfMilliseconds milliseconds to AValue and returns the resulting date/time. ANumberOfMilliseconds can be positive or negative.

See also: IncYear (361), #rtl.sysutils.IncMonth (1154), IncWeek (360), IncDay (358), IncHour (358), IncSecond (360), IncMilliSecond (359)

Listing: ./datutex/ex78.pp

Program Example78;

{ This program demonstrates the IncMilliSecond function }

Uses SysUtils, DateUtils;

Begin

WriteLn ('One MilliSecond from now is ', **TimeToStr**(IncMilliSecond(**Now**, 1)));

WriteLn ('One MilliSecond ago from now is ', **TimeToStr**(IncMilliSecond(**Now**, -1)));

End.

IncMinute

Synopsis: Increase a DateTime value with a number of minutes.

Declaration: `function IncMinute(const AValue: TDateTime;
const ANumberOfMinutes: Int64) : TDateTime
function IncMinute(const AValue: TDateTime) : TDateTime`

Visibility: default

Description: IncMinute adds ANumberOfMinutes minutes to AValue and returns the resulting date/-time. ANumberOfMinutes can be positive or negative.

See also: IncYear (361), #rtl.sysutils.IncMonth (1154), IncWeek (360), IncDay (358), IncHour (358), IncSecond (360), IncMilliSecond (359)

Listing: ./datutex/ex76.pp

Program Example76;

{ This program demonstrates the IncMinute function }

Uses SysUtils, DateUtils;

Begin

WriteLn ('One Minute from now is ', **TimeToStr**(IncMinute(**Time**, 1)));

WriteLn ('One Minute ago from now is ', **TimeToStr**(IncMinute(**Time**, -1)));

End.

IncSecond

Synopsis: Increase a DateTime value with a number of seconds.

Declaration: `function IncSecond(const AValue: TDateTime;
 const ANumberOfSeconds: Int64) : TDateTime
 function IncSecond(const AValue: TDateTime) : TDateTime`

Visibility: default

Description: IncSecond adds ANumberOfSeconds seconds to AValue and returns the resulting date/-time. ANumberOfSeconds can be positive or negative.

See also: IncYear (361), #rtl.sysutils.IncMonth (1154), IncWeek (360), IncDay (358), IncHour (358), IncSecond (360), IncMilliSecond (359)

Listing: ./datutex/ex77.pp

Program Example77;

{ This program demonstrates the IncSecond function }

Uses SysUtils, DateUtils;

Begin

WriteIn('One Second from now is ', **TimeToStr**(IncSecond(**Time**, 1)));

WriteIn('One Second ago from now is ', **TimeToStr**(IncSecond(**Time**, -1)));

End.

IncWeek

Synopsis: Increase a DateTime value with a number of weeks.

Declaration: `function IncWeek(const AValue: TDateTime; const ANumberOfWeeks: Integer)
 : TDateTime
 function IncWeek(const AValue: TDateTime) : TDateTime`

Visibility: default

Description: IncWeek adds ANumberOfWeeks weeks to AValue and returns the resulting date/time. ANumberOfWeeks can be positive or negative.

See also: IncYear (361), #rtl.sysutils.IncMonth (1154), IncDay (358), IncHour (358), IncMinute (359), IncSecond (360), IncMilliSecond (359)

Listing: ./datutex/ex73.pp

Program Example73;

{ This program demonstrates the IncWeek function }

Uses SysUtils, DateUtils;

Begin

WriteIn('One Week from today is ', **DateToStr**(IncWeek(**Today**, 1)));

WriteIn('One Week ago from today is ', **DateToStr**(IncWeek(**Today**, -1)));

End.

IncYear

Synopsis: Increase a `DateTime` value with a number of years.

Declaration: `function IncYear(const AValue: TDateTime;const ANumberOfYears: Integer)
: TDateTime
function IncYear(const AValue: TDateTime) : TDateTime`

Visibility: default

Description: `IncYear` adds `ANumberOfYears` years to `AValue` and returns the resulting date/time. `ANumberOfYears` can be positive or negative.

See also: `#rtl.sysutils.IncMonth` (1154), `IncWeek` (360), `IncDay` (358), `IncHour` (358), `IncMinute` (359), `IncSecond` (360), `IncMilliSecond` (359)

Listing: `./datutex/ex71.pp`

Program `Example71;`

`{ This program demonstrates the IncYear function }`

Uses `SysUtils, DateUtils;`

Begin

`WriteLn('One year from today is ',DateToStr(IncYear(Today,1)));`

`WriteLn('One year ago from today is ',DateToStr(IncYear(Today,-1)));`

End.

InvalidDateDayError

Synopsis: Raise an `EConvertError` exception when a day is not a valid day of a year.

Declaration: `procedure InvalidDateDayError(const AYear: Word;const ADayOfYear: Word)`

Visibility: default

Description: `InvalidDateDayError` raises an `EConvertError` (1194) exception and formats the error message with an appropriate description made up from the parts `AYear` and `ADayOfYear`.

Normally this function should not be needed, the conversion routines call it when they have received invalid arguments.

See also: `InvalidDateWeekError` (362), `InvalidDateTimeError` (362), `InvalidDateMonthWeekError` (361), `InvalidDayOfWeekInMonthError` (363)

InvalidDateMonthWeekError

Synopsis: Raise an `EConvertError` exception when a `Year,Month,WeekOfMonth,DayOfWeek` is invalid.

Declaration: `procedure InvalidDateMonthWeekError(const AYear: Word;
const AMonth: Word;
const AWeekOfMonth: Word;
const ADayOfWeek: Word)`

Visibility: default

Description: `InvalidDateMonthWeekError` raises an `EConvertError` (1194) exception and formats the error message with an appropriate description made up from the parts `AYear`, `AMonth`, `AWeekOfMonth` and `ADayOfWeek`.

Normally this function should not be needed, the conversion routines call it when they have received invalid arguments.

See also: `InvalidDateWeekError` (362), `InvalidDateTimeError` (362), `InvalidDateDayError` (361), `InvalidDay-Of-WeekInMonthError` (363)

InvalidDateTimeError

Synopsis: Raise an `EConvertError` about an invalid date-time specification.

Declaration:

```
procedure InvalidDateTimeError(const AYear: Word;const AMonth: Word;
                               const ADay: Word;const AHour: Word;
                               const AMinute: Word;const ASecond: Word;
                               const AMilliSecond: Word;
                               const ABaseDate: TDateTime)
procedure InvalidDateTimeError(const AYear: Word;const AMonth: Word;
                               const ADay: Word;const AHour: Word;
                               const AMinute: Word;const ASecond: Word;
                               const AMilliSecond: Word)
```

Visibility: default

Description: `InvalidDateTimeError` raises an `EConvertError` (1194) exception and formats the error message with an appropriate description made up from the parts `AYear`, `AMonth`, `ADay`, `AHour`, `AMinute`, `ASecond` and `AMilliSecond`.

Normally this function should not be needed, the conversion routines call it when they have received invalid arguments.

See also: `InvalidDateWeekError` (362), `InvalidDateDayError` (361), `InvalidDateMonthWeekError` (361), `InvalidDayOf-WeekInMonthError` (363)

InvalidDateWeekError

Synopsis: Raise an `EConvertError` with an invalid Year, WeekOfyear and DayOfWeek specification

Declaration:

```
procedure InvalidDateWeekError(const AYear: Word;
                               const AWeekOfYear: Word;
                               const ADayOfWeek: Word)
```

Visibility: default

Description: `InvalidDateWeekError` raises an `EConvertError` (1194) exception and formats the error message with an appropriate description made up from the parts `AYear`, `AWeek`, `ADayOfWeek`

Normally this function should not be needed, the conversion routines call it when they have received invalid arguments.

See also: `InvalidDateTimeError` (362), `InvalidDateDayError` (361), `InvalidDateMonthWeekError` (361), `InvalidDayOf-WeekInMonthError` (363)

InvalidDayOfWeekInMonthError

Synopsis: Raise an `EConvertError` exception when a `Year,Month,NthDayOfWeek,DayOfWeek` is invalid.

Declaration: `procedure InvalidDayOfWeekInMonthError(const AYear: Word;
const AMonth: Word;
const ANthDayOfWeek: Word;
const ADayOfWeek: Word)`

Visibility: default

Description: `InvalidDayOfWeekInMonthError` raises an `EConvertError` (1194) exception and formats the error message with an appropriate description made up from the parts `AYear`, `AMonth`, `ANthDayOfWeek` and `ADayOfWeek`.

Normally this function should not be needed, the conversion routines call it when they have received invalid arguments.

See also: `InvalidDateWeekError` (362), `InvalidDateTimeError` (362), `InvalidDateDayError` (361), `InvalidDate-MonthWeekError` (361)

IsInLeapYear

Synopsis: Determine whether a date is in a leap year.

Declaration: `function IsInLeapYear(const AValue: TDateTime) : Boolean`

Visibility: default

Description: `IsInLeapYear` returns `True` if the year part of `AValue` is leap year, or `False` if not.

See also: `YearOf` (417), `IsPM` (363), `IsToday` (364), `IsSameDay` (364)

Listing: `./datutex/ex3.pp`

Program Example3;

{ This program demonstrates the IsInLeapYear function }

Uses SysUtils, DateUtils;

Begin

WriteIn ('Current year is leap year: ', IsInLeapYear(Date));

End.

IsPM

Synopsis: Determine whether a time is PM or AM.

Declaration: `function IsPM(const AValue: TDateTime) : Boolean`

Visibility: default

Description: `IsPM` returns `True` if the time part of `AValue` is later then 12:00 (PM, or afternoon).

See also: `YearOf` (417), `IsInLeapYear` (363), `IsToday` (364), `IsSameDay` (364)

Listing: `./datutex/ex4.pp`

```

Program Example4;

{ This program demonstrates the IsPM function }

Uses SysUtils, DateUtils;

Begin
    WriteLn('Current time is PM : ', IsPM(Now));
End.

```

IsSameDay

Synopsis: Check if two date/time indications are the same day.

Declaration: `function IsSameDay(const AValue: TDateTime; const ABasis: TDateTime) : Boolean`

Visibility: default

Description: `IsSameDay` checks whether `AValue` and `ABasis` have the same date part, and returns `True` if they do, `False` if not.

See also: [Today \(399\)](#), [Yesterday \(420\)](#), [Tomorrow \(399\)](#), [IsToday \(364\)](#)

Listing: ./datutex/ex21.pp

```

Program Example21;

{ This program demonstrates the IsSameDay function }

Uses SysUtils, DateUtils;

Var
    I : Integer;
    D : TDateTime;

Begin
    For I:=1 to 3 do
        begin
            D:=Today+Random(3)-1;
            Write(FormatDateTime('dd mmm yyyy "is today : "', D));
            WriteLn(IsSameDay(D, Today));
        end;
End.

```

IsToday

Synopsis: Check whether a given date is today.

Declaration: `function IsToday(const AValue: TDateTime) : Boolean`

Visibility: default

Description: `IsToday` returns `True` if `AValue` is today's date, and `False` otherwise.

See also: [Today \(399\)](#), [Yesterday \(420\)](#), [Tomorrow \(399\)](#), [IsSameDay \(364\)](#)

Listing: ./datutex/ex20.pp

Program Example20;

{ This program demonstrates the IsToday function }

Uses SysUtils, DateUtils;

Begin

```
WriteLn('Today      : ', IsToday(Today));
WriteLn('Tomorrow   : ', IsToday(Tomorrow));
WriteLn('Yesterday  : ', IsToday(Yesterday));
```

End.

IsValidDate

Synopsis: Check whether a set of values is a valid date indication.

Declaration: function IsValidDate(const AYear: Word; const AMonth: Word;
const ADay: Word) : Boolean

Visibility: default

Description: IsValidDate returns True when the values AYear, AMonth, ADay form a valid date indication. If one of the values is not valid (e.g. the day is invalid or does not exist in that particular month), False is returned.

AYear must be in the range 1..9999 to be valid.

See also: IsValidTime ([369](#)), IsValidDateTime ([367](#)), IsValidDateDay ([365](#)), IsValidDateWeek ([368](#)), IsValidDateMonthWeek ([366](#))

Listing: ./datutex/ex5.pp

Program Example5;

{ This program demonstrates the IsValidDate function }

Uses SysUtils, DateUtils;

Var

Y,M,D : Word;

Begin

```
For Y:=2000 to 2004 do
  For M:=1 to 12 do
    For D:=1 to 31 do
      If Not IsValidDate(Y,M,D) then
        WriteLn(D, ' is not a valid day in ', Y, '/', M);
```

End.

IsValidDateDay

Synopsis: Check whether a given year/day of year combination is a valid date.

Declaration: function IsValidDateDay(const AYear: Word; const ADayOfYear: Word)
: Boolean

Visibility: default

Description: IsValidDateDay returns True if AYear and ADayOfYear form a valid date indication, or False otherwise.

AYear must be in the range 1..9999 to be valid.

The ADayOfYear value is checked to see whether it falls within the valid range of dates for AYear.

See also: IsValidDate (365), IsValidTime (369), IsValidDateTime (367), IsValidDateWeek (368), IsValidDateMonthWeek (366)

Listing: ./datutex/ex9.pp

Program Example9;

{ This program demonstrates the IsValidDateDay function }

Uses SysUtils , DateUtils ;

Var

Y : Word;

Begin

For Y:=1996 to 2004 do

if IsValidDateDay(Y,366) then

WriteLn(Y, ' is a leap year');

End.

IsValidDateMonthWeek

Synopsis: Check whether a given year/month/week/day of the week combination is a valid day

Declaration: function IsValidDateMonthWeek(const AYear: Word;const AMonth: Word;
const AWeekOfMonth: Word;
const ADayOfWeek: Word) : Boolean

Visibility: default

Description: IsValidDateMonthWeek returns True if AYear, AMonth, AWeekOfMonth and ADayOfWeek form a valid date indication, or False otherwise.

AYear must be in the range 1..9999 to be valid.

The AWeekOfMonth,ADayOfWeek values are checked to see whether the combination falls within the valid range of weeks for the AYear,AMonth combination.

See also: IsValidDate (365), IsValidTime (369), IsValidDateTime (367), IsValidDateDay (365), IsValidDateWeek (368)

Listing: ./datutex/ex11.pp

Program Example11;

{ This program demonstrates the IsValidDateMonthWeek function }

Uses SysUtils , DateUtils ;

Var

Y,W,D : Word;

```

    B : Boolean;

Begin
  For Y:=2000 to 2004 do
    begin
      B:=True;
      For W:=4 to 6 do
        For D:=1 to 7 do
          If B then
            begin
              B:=IsValidDateMonthWeek(Y,12,W,D);
            end
          If Not B then
            if (D=1) then
              Writeln('December ',Y,' has exactly ',W,' weeks. ');
            else
              Writeln('December ',Y,' has ',W,' weeks and ',D-1,' days. ');
            end;
          end;
        end;
      end;
    end;
  End.

```

IsValidDateTime

Synopsis: Check whether a set of values is a valid date and time indication.

Declaration: function IsValidDateTime(const AYear: Word;const AMonth: Word;
 const ADay: Word;const AHour: Word;
 const AMinute: Word;const ASecond: Word;
 const AMilliSecond: Word) : Boolean

Visibility: default

Description: IsValidTime returns True when the values AYear, AMonth, ADay, AHour, AMinute, ASecond and AMilliSecond form a valid date and time indication. If one of the values is not valid (e.g. the seconds are larger than 60), False is returned.

AYear must be in the range 1..9999 to be valid.

See also: IsValidDate ([365](#)), IsValidTime ([369](#)), IsValidDateDay ([365](#)), IsValidDateWeek ([368](#)), IsValidDateMonthWeek ([366](#))

Listing: ./datutex/ex7.pp

Program Example7;

{ This program demonstrates the IsValidDateTime function }

Uses SysUtils, DateUtils;

Var

```

  Y,Mo,D : Word;
  H,M,S,MS : Word;
  I : Integer;

```

Begin

```

  For I:=1 to 10 do
    begin
      Y:=2000+Random(5);
      Mo:=Random(15);

```

```

D:=Random(40);
H:=Random(32);
M:=Random(90);
S:=Random(90);
MS:=Random(1500);
If Not IsValidDateTime(Y,Mo,D,H,M,S,MS) then
  WriteLn(Y,'-',Mo,'-',D,' ',H,':',M,':',S,'.',MS,' is not a valid date/time. ');
end;
End.

```

IsValidDateWeek

Synopsis: Check whether a given year/week/day of the week combination is a valid day.

Declaration: `function IsValidDateWeek(const AYear: Word;const AWeekOfYear: Word;
const ADayOfWeek: Word) : Boolean`

Visibility: default

Description: IsValidDateWeek returns True if AYear, AWeekOfYear and ADayOfWeek form a valid date indication, or False otherwise.

AYear must be in the range 1..9999 to be valid.

The ADayOfWeek,ADayOfWeek values are checked to see whether the combination falls within the valid range of weeks for AYear.

See also: IsValidDate (365), IsValidTime (369), IsValidDateTime (367), IsValidDateDay (365), IsValidDate-MonthWeek (366)

Listing: ./datutex/ex10.pp

Program Example10;

{ This program demonstrates the IsValidDateWeek function }

Uses SysUtils, DateUtils;

Var

Y,W,D : Word;
B : Boolean;

Begin

```

For Y:=2000 to 2004 do
  begin
    B:=True;
    For W:=51 to 54 do
      For D:=1 to 7 do
        If B then
          begin
            B:=IsValidDateWeek(Y,W,D);
            If Not B then
              if (D=1) then
                WriteLn(Y,' has exactly ',W,' weeks. ');
              else
                WriteLn(Y,' has ',W,' weeks and ',D-1,' days. ');
            end;
          end;
        end;
      end;
    end;
  end;
end;

```

End.

IsValidTime

Synopsis: Check whether a set of values is a valid time indication.

Declaration: `function IsValidTime(const AHour: Word;const AMinute: Word;
const ASecond: Word;const AMilliSecond: Word)
: Boolean`

Visibility: default

Description: Check whether a set of values is a valid time indication.

JulianDateToDateTime

Synopsis: Convert a Julian date representation to a TDateTime value.

Declaration: `function JulianDateToDateTime(const AValue: Double) : TDateTime`

Visibility: default

Description: Not yet implemented.

Errors: Currently, trying to use this function will raise an exception.

See also: [DateTimeToJulianDate \(339\)](#), [TryJulianDateToDateTime \(403\)](#), [DateTimeToModifiedJulianDate \(339\)](#), [TryModifiedJulianDateToDateTime \(403\)](#)

MilliSecondOf

Synopsis: Extract the millisecond part from a DateTime value.

Declaration: `function MilliSecondOf(const AValue: TDateTime) : Word`

Visibility: default

Description: `MilliSecondOf` returns the second of the minute part of the AValue date/time indication. It is a number between 0 and 999.

For an example, see [YearOf \(417\)](#)

See also: [YearOf \(417\)](#), [WeekOf \(405\)](#), [MonthOf \(377\)](#), [DayOf \(339\)](#), [HourOf \(354\)](#), [MinuteOf \(373\)](#), [MilliSecondOf \(369\)](#)

MilliSecondOfDay

Synopsis: Calculate the number of milliseconds elapsed since the start of the day

Declaration: `function MilliSecondOfDay(const AValue: TDateTime) : LongWord`

Visibility: default

Description: `MilliSecondOfDay` returns the number of milliseconds that have passed since the start of the Day (00:00:00.000) till the moment indicated by AValue. This is a zero-based number, i.e. 00:00:00.000 will return 0.

For an example, see the [HourOfDay \(354\)](#) function.

See also: [MilliSecondOfTheYear \(371\)](#), [MilliSecondOfTheMonth \(370\)](#), [MilliSecondOfTheWeek \(371\)](#), [MilliSecondOfTheHour \(370\)](#), [MilliSecondOfTheMinute \(370\)](#), [MilliSecondOfTheSecond \(370\)](#), [HourOfTheDay \(354\)](#), [MinuteOfDay \(373\)](#), [SecondOfDay \(390\)](#)

MilliSecondOfTheHour

Synopsis: Calculate the number of milliseconds elapsed since the start of the hour

Declaration: `function MilliSecondOfTheHour(const AValue: TDateTime) : LongWord`

Visibility: default

Description: `MilliSecondOfTheHour` returns the number of milliseconds that have passed since the start of the Hour (HH:00:00.000) till the moment indicated by `AValue`. This is a zero-based number, i.e. HH:00:00.000 will return 0.

For an example, see the `MinuteOfTheHour` (374) function.

See also: `MilliSecondOfTheYear` (371), `MilliSecondOfTheMonth` (370), `MilliSecondOfTheWeek` (371), `MilliSecondOfTheDay` (369), `MilliSecondOfTheMinute` (370), `MilliSecondOfTheSecond` (370), `MinuteOfTheHour` (374), `SecondOfTheHour` (390)

MilliSecondOfTheMinute

Synopsis: Calculate the number of milliseconds elapsed since the start of the minute

Declaration: `function MilliSecondOfTheMinute(const AValue: TDateTime) : LongWord`

Visibility: default

Description: `MilliSecondOfTheMinute` returns the number of milliseconds that have passed since the start of the Minute (HH:MM:00.000) till the moment indicated by `AValue`. This is a zero-based number, i.e. HH:MM:00.000 will return 0.

For an example, see the `SecondOfTheMinute` (391) function.

See also: `MilliSecondOfTheYear` (371), `MilliSecondOfTheMonth` (370), `MilliSecondOfTheWeek` (371), `MilliSecondOfTheDay` (369), `MilliSecondOfTheHour` (370), `MilliSecondOfTheMinute` (370), `MilliSecondOfTheSecond` (370), `SecondOfTheMinute` (391)

MilliSecondOfTheMonth

Synopsis: Calculate number of milliseconds elapsed since the start of the month.

Declaration: `function MilliSecondOfTheMonth(const AValue: TDateTime) : LongWord`

Visibility: default

Description: `MilliSecondOfTheMonth` returns the number of milliseconds that have passed since the start of the month (00:00:00.000) till the moment indicated by `AValue`. This is a zero-based number, i.e. 00:00:00.000 on the first of the month will return 0.

For an example, see the `WeekOfTheMonth` (405) function.

See also: `WeekOfTheMonth` (405), `DayOfTheMonth` (340), `HourOfTheMonth` (355), `MinuteOfTheMonth` (374), `SecondOfTheMonth` (391), `MilliSecondOfTheMonth` (370)

MilliSecondOfTheSecond

Synopsis: Calculate the number of milliseconds elapsed since the start of the second

Declaration: `function MilliSecondOfTheSecond(const AValue: TDateTime) : Word`

Visibility: default

Description: `MilliSecondOfTheSecond` returns the number of milliseconds that have passed since the start of the second (HH:MM:SS.000) till the moment indicated by `AValue`. This is a zero-based number, i.e. HH:MM:SS.000 will return 0.

See also: `MilliSecondOfTheYear` (371), `MilliSecondOfTheMonth` (370), `MilliSecondOfTheWeek` (371), `MilliSecondOfTheDay` (369), `MilliSecondOfTheHour` (370), `MilliSecondOfTheMinute` (370), `SecondOfTheMinute` (391)

Listing: `./datutex/ex46.pp`

Program `Example46`;

{ This program demonstrates the MilliSecondOfTheSecond function }

Uses `SysUtils` , `DateUtils` ;

Var

`N` : `TDateTime` ;

Begin

`N:=Now`;

`WriteLn` (' MilliSecond of the Second : ' ,
 `MilliSecondOfTheSecond` (`N`));

End.

MilliSecondOfTheWeek

Synopsis: Calculate the number of milliseconds elapsed since the start of the week

Declaration: `function MilliSecondOfTheWeek(const AValue: TDateTime) : LongWord`

Visibility: `default`

Description: `MilliSecondOfTheWeek` returns the number of milliseconds that have passed since the start of the Week (00:00:00.000) till the moment indicated by `AValue`. This is a zero-based number, i.e. 00:00:00.000 on the first of the Week will return 0.

For an example, see the `DayOfTheWeek` (340) function.

See also: `MilliSecondOfTheYear` (371), `MilliSecondOfTheMonth` (370), `MilliSecondOfTheDay` (369), `MilliSecondOfTheHour` (370), `MilliSecondOfTheMinute` (370), `MilliSecondOfTheSecond` (370), `DayOfTheWeek` (340), `HourOfTheWeek` (355), `MinuteOfTheWeek` (375), `SecondOfTheWeek` (391)

MilliSecondOfTheYear

Synopsis: Calculate the number of milliseconds elapsed since the start of the year.

Declaration: `function MilliSecondOfTheYear(const AValue: TDateTime) : Int64`

Visibility: `default`

Description: `MilliSecondOfTheYear` returns the number of milliseconds that have passed since the start of the year (January 1, 00:00:00.000) till the moment indicated by `AValue`. This is a zero-based number, i.e. January 1 00:00:00.000 will return 0.

For an example, see the `WeekOfTheYear` (406) function.

See also: `WeekOfTheYear` (406), `DayOfTheYear` (341), `HourOfTheYear` (356), `MinuteOfTheYear` (375), `SecondOfTheYear` (392), `MilliSecondOfTheYear` (371)

MillisecondsBetween

Synopsis: Calculate the number of whole milliseconds between two DateTime values.

Declaration: `function MillisecondsBetween(const ANow: TDateTime;
const AThen: TDateTime) : Int64`

Visibility: default

Description: `MillisecondsBetween` returns the number of whole milliseconds between `ANow` and `AThen`. This means a fractional part of a millisecond is dropped.

See also: `YearsBetween` (418), `MonthsBetween` (378), `WeeksBetween` (407), `DaysBetween` (341), `HoursBetween` (356), `MinutesBetween` (375), `SecondsBetween` (392)

Listing: `./datutex/ex62.pp`

Program Example62;

{ This program demonstrates the MillisecondsBetween function }

Uses SysUtils, DateUtils;

Procedure Test(ANow, AThen : TDateTime);

begin

 Write('Number of milliseconds between ');
 Write(TimeToStr(AThen), ' and ', TimeToStr(ANow));
 WriteLn(' : ', MillisecondsBetween(ANow, AThen));
end;

Var

 D1, D2 : TDateTime;

Begin

 D1:=Now;
 D2:=D1-(0.9*OneMilliSecond);
 Test(D1,D2);
 D2:=D1-(1.0*OneMilliSecond);
 Test(D1,D2);
 D2:=D1-(1.1*OneMilliSecond);
 Test(D1,D2);
 D2:=D1-(2.5*OneMilliSecond);
 Test(D1,D2);

End.

MilliSecondSpan

Synopsis: Calculate the approximate number of milliseconds between two DateTime values.

Declaration: `function MilliSecondSpan(const ANow: TDateTime; const AThen: TDateTime)
: Double`

Visibility: default

Description: `MilliSecondSpan` returns the number of milliseconds between `ANow` and `AThen`. Since millisecond is the smallest fraction of a `TDateTime` indication, the returned number will always be an integer value.

See also: [YearSpan \(419\)](#), [MonthSpan \(379\)](#), [WeekSpan \(409\)](#), [DaySpan \(344\)](#), [HourSpan \(357\)](#), [MinuteSpan \(376\)](#), [SecondSpan \(393\)](#), [MillisecondsBetween \(372\)](#)

Listing: ./datutex/ex70.pp

Program Example70;

{ This program demonstrates the MilliSecondSpan function }

Uses SysUtils, DateUtils;

Procedure Test(ANow, AThen : TDateTime);

begin

Write('Number of milliseconds between ');
 Write(**TimeToStr**(AThen), ' and ', **TimeToStr**(ANow));
 WriteLn(' : ', MilliSecondSpan(ANow, AThen));
end;

Var

 D1, D2 : TDateTime;

Begin

 D1:=**Now**;
 D2:=D1-(0.9*OneMilliSecond);
 Test(D1,D2);
 D2:=D1-(1.0*OneMilliSecond);
 Test(D1,D2);
 D2:=D1-(1.1*OneMilliSecond);
 Test(D1,D2);
 D2:=D1-(2.5*OneMilliSecond);
 Test(D1,D2);

End.

MinuteOf

Synopsis: Extract the minute part from a DateTime value.

Declaration: function MinuteOf(const AValue: TDateTime) : Word

Visibility: default

Description: MinuteOf returns the minute of the hour part of the AValue date/time indication. It is a number between 0 and 59.

For an example, see [YearOf \(417\)](#)

See also: [YearOf \(417\)](#), [WeekOf \(405\)](#), [MonthOf \(377\)](#), [DayOf \(339\)](#), [HourOf \(354\)](#), [SecondOf \(390\)](#), [MilliSecondOf \(369\)](#)

MinuteOfDay

Synopsis: Calculate the number of minutes elapsed since the start of the day

Declaration: function MinuteOfDay(const AValue: TDateTime) : Word

Visibility: default

Description: `MinuteOfDay` returns the number of minutes that have passed since the start of the Day (00:00:00) till the moment indicated by `AValue`. This is a zero-based number, i.e. 00:00:59 will return 0.

For an example, see the `HourOfDay` (354) function.

See also: `MinuteOfYear` (375), `MinuteOfMonth` (374), `MinuteOfWeek` (375), `MinuteOfTheHour` (374), `HourOfDay` (354), `SecondOfDay` (390), `MilliSecondOfDay` (369)

MinuteOfTheHour

Synopsis: Calculate the number of minutes elapsed since the start of the hour

Declaration: `function MinuteOfTheHour(const AValue: TDateTime) : Word`

Visibility: default

Description: `MinuteOfTheHour` returns the number of minutes that have passed since the start of the Hour (HH:00:00) till the moment indicated by `AValue`. This is a zero-based number, i.e. HH:00:59 will return 0.

See also: `MinuteOfYear` (375), `MinuteOfMonth` (374), `MinuteOfWeek` (375), `MinuteOfDay` (373), `SecondOfTheHour` (390), `MilliSecondOfTheHour` (370)

Listing: `./datutex/ex44.pp`

Program Example44;

{ This program demonstrates the MinuteOfTheHour function }

Uses SysUtils, DateUtils;

Var

N : TDateTime;

Begin

N:=Now;

WriteLn('Minute of the Hour : ',MinuteOfTheHour(N));

WriteLn('Second of the Hour : ',SecondOfTheHour(N));

WriteLn('MilliSecond of the Hour : ',
MilliSecondOfTheHour(N));

End.

MinuteOfMonth

Synopsis: Calculate number of minutes elapsed since the start of the month.

Declaration: `function MinuteOfMonth(const AValue: TDateTime) : Word`

Visibility: default

Description: `MinuteOfMonth` returns the number of minutes that have passed since the start of the Month (00:00:00) till the moment indicated by `AValue`. This is a zero-based number, i.e. 00:00:59 on the first day of the month will return 0.

For an example, see the `WeekOfMonth` (405) function.

See also: `WeekOfMonth` (405), `DayOfMonth` (340), `HourOfMonth` (355), `MinuteOfMonth` (374), `SecondOfMonth` (391), `MilliSecondOfMonth` (370)

MinuteOfTheWeek

Synopsis: Calculate the number of minutes elapsed since the start of the week

Declaration: `function MinuteOfTheWeek(const AValue: TDateTime) : Word`

Visibility: default

Description: `MinuteOfTheWeek` returns the number of minutes that have passed since the start of the week (00:00:00) till the moment indicated by `AValue`. This is a zero-based number, i.e. 00:00:59 on the first day of the week will return 0.

For an example, see the `DayOfTheWeek` (340) function.

See also: `MinuteOfTheYear` (375), `MinuteOfTheMonth` (374), `MinuteOfTheDay` (373), `MinuteOfTheHour` (374), `DayOfTheWeek` (340), `HourOfTheWeek` (355), `SecondOfTheWeek` (391), `MilliSecondOfTheWeek` (371)

MinuteOfTheYear

Synopsis: Calculate the number of minutes elapsed since the start of the year

Declaration: `function MinuteOfTheYear(const AValue: TDateTime) : LongWord`

Visibility: default

Description: `MinuteOfTheYear` returns the number of minutes that have passed since the start of the year (January 1, 00:00:00) till the moment indicated by `AValue`. This is a zero-based number, i.e. January 1 00:00:59 will return 0.

For an example, see the `WeekOfTheYear` (406) function.

See also: `WeekOfTheYear` (406), `DayOfTheYear` (341), `HourOfTheYear` (356), `MinuteOfTheYear` (375), `SecondOfTheYear` (392), `MilliSecondOfTheYear` (371)

MinutesBetween

Synopsis: Calculate the number of whole minutes between two `DateTime` values.

Declaration: `function MinutesBetween(const ANow: TDateTime;const AThen: TDateTime) : Int64`

Visibility: default

Description: `MinutesBetween` returns the number of whole minutes between `ANow` and `AThen`. This means the fractional part of a minute (seconds, milliseconds etc.) is dropped.

See also: `YearsBetween` (418), `MonthsBetween` (378), `WeeksBetween` (407), `DaysBetween` (341), `HoursBetween` (356), `SecondsBetween` (392), `MillisecondsBetween` (372)

Listing: `./datutex/ex60.pp`

Program `Example60;`

{ This program demonstrates the MinutesBetween function }

Uses `SysUtils, DateUtils;`

Procedure `Test(ANow, AThen : TDateTime);`

```

begin
  Write( 'Number of minutes between ');
  Write( TimeToStr(AThen), ' and ', TimeToStr(ANow));
  WriteLn( ' : ', MinutesBetween(ANow, AThen));
end;

Var
  D1, D2 : TDateTime;

Begin
  D1:=Now;
  D2:=D1-(59*OneSecond);
  Test(D1,D2);
  D2:=D1-(61*OneSecond);
  Test(D1,D2);
  D2:=D1-(122*OneSecond);
  Test(D1,D2);
  D2:=D1-(306*OneSecond);
  Test(D1,D2);
  D2:=D1-(5.4*OneMinute);
  Test(D1,D2);
  D2:=D1-(2.5*OneMinute);
  Test(D1,D2);
End.

```

MinuteSpan

Synopsis: Calculate the approximate number of minutes between two DateTime values.

Declaration: `function MinuteSpan(const ANow: TDateTime; const AThen: TDateTime) : Double`

Visibility: default

Description: MinuteSpan returns the number of minutes between ANow and AThen, including any fractional parts of a minute.

See also: YearSpan ([419](#)), MonthSpan ([379](#)), WeekSpan ([409](#)), DaySpan ([344](#)), HourSpan ([357](#)), SecondSpan ([393](#)), MilliSecondSpan ([372](#)), MinutesBetween ([375](#))

Listing: ./datutex/ex68.pp

Program Example68;

```
{ This program demonstrates the MinuteSpan function }
```

Uses SysUtils, DateUtils;

Procedure Test(ANow, AThen : TDateTime);

```

begin
  Write( 'Number of minutes between ');
  Write( TimeToStr(AThen), ' and ', TimeToStr(ANow));
  WriteLn( ' : ', MinuteSpan(ANow, AThen));
end;

```

Var

D1,D2 : TDateTime;

Begin

```
D1:=Now;
D2:=D1-(59*OneSecond);
Test(D1,D2);
D2:=D1-(61*OneSecond);
Test(D1,D2);
D2:=D1-(122*OneSecond);
Test(D1,D2);
D2:=D1-(306*OneSecond);
Test(D1,D2);
D2:=D1-(5.4*OneMinute);
Test(D1,D2);
D2:=D1-(2.5*OneMinute);
Test(D1,D2);
```

End.

ModifiedJulianDateToDateTime

Synopsis: Convert a modified Julian date representation to a TDateTime value.

Declaration: `function ModifiedJulianDateToDateTime(const AValue: Double) : TDateTime`

Visibility: default

Description: Not yet implemented.

Errors: Currently, trying to use this function will raise an exception.

See also: [DateTimeToJulianDate \(339\)](#), [JulianDateToDateTime \(369\)](#), [TryJulianDateToDateTime \(403\)](#), [DateTimeToModifiedJulianDate \(339\)](#), [TryModifiedJulianDateToDateTime \(403\)](#)

MonthOf

Synopsis: Extract the month from a given date.

Declaration: `function MonthOf(const AValue: TDateTime) : Word`

Visibility: default

Description: `MonthOf` returns the month part of the `AValue` date/time indication. It is a number between 1 and 12.

For an example, see [YearOf \(417\)](#)

See also: [YearOf \(417\)](#), [DayOf \(339\)](#), [WeekOf \(405\)](#), [HourOf \(354\)](#), [MinuteOf \(373\)](#), [SecondOf \(390\)](#), [MilliSecondOf \(369\)](#)

MonthOfTheYear

Synopsis: Extract the month of a DateTime indication.

Declaration: `function MonthOfTheYear(const AValue: TDateTime) : Word`

Visibility: default

Description: `MonthOfTheYear` extracts the month part of `Avalue` and returns it. It is an alias for `MonthOf` (377), and is provided for completeness only, corresponding to the other `PartOfTheYear` functions.

For an example, see the [WeekOfTheYear \(406\)](#) function.

See also: MonthOf ([377](#)), WeekOfTheYear ([406](#)), DayOfTheYear ([341](#)), HourOfTheYear ([356](#)), MinuteOfTheYear ([375](#)), SecondOfTheYear ([392](#)), MilliSecondOfTheYear ([371](#))

MonthsBetween

Synopsis: Calculate the number of whole months between two DateTime values

```
Declaration: function MonthsBetween(const ANow: TDateTime;const AThen: TDateTime)
                                : Integer
```

Visibility: default

Description: MonthsBetween returns the number of whole months between ANow and AThen. This number is an approximation, based on an average number of days of 30.4375 per month (average over 4 years). This means the fractional part of a month is dropped.

See also: [YearsBetween \(418\)](#), [WeeksBetween \(407\)](#), [DaysBetween \(341\)](#), [HoursBetween \(356\)](#), [MinutesBetween \(375\)](#), [SecondsBetween \(392\)](#), [MilliSecondsBetween \(372\)](#)

Listing: ./datutex/ex56.pp

Program Example56;

```
{ This program demonstrates the MonthsBetween function }
```

Uses SysUtils , DateUtils ;

```
Procedure Test (ANow, AThen : TDateTime);
```

begin

```

Write ('Number of months between ');
Write (DateToStr (AThen), ' and ', DateToStr (ANow));
WriteIn (' : ', MonthsBetween (ANow, AThen));
end;

```

Var

D1,D2 : TDateTime;

Begin

```
D1:=Today ;
D2:=Today-364;
Test (D1,D2) ;
D2:=Today-365;
Test (D1,D2) ;
D2:=Today-366;
Test (D1,D2) ;
D2:=Today-390;
Test (D1,D2) ;
D2:=Today-368;
Test (D1,D2) ;
D2:=Today-1000;
Test (D1,D2) ;
```

End.

MonthSpan

Synopsis: Calculate the approximate number of months between two DateTime values.

Declaration: `function MonthSpan(const ANow: TDateTime;const AThen: TDateTime)
: Double`

Visibility: default

Description: MonthSpan returns the number of month between ANow and AThen, including any fractional parts of a month. This number is an approximation, based on an average number of days of 30.4375 per month (average over 4 years).

See also: YearSpan ([419](#)), WeekSpan ([409](#)), DaySpan ([344](#)), HourSpan ([357](#)), MinuteSpan ([376](#)), SecondSpan ([393](#)), MilliSecondSpan ([372](#)), MonthsBetween ([378](#))

Listing: ./datutex/ex64.pp

Program Example64;

{ This program demonstrates the MonthSpan function }

Uses SysUtils, DateUtils;

Procedure Test(ANow, AThen : TDateTime);

begin

 Write('Number of months between ');

 Write(DateToStr(AThen), ' and ', DateToStr(ANow));

 WriteLn(' : ', MonthSpan(ANow, AThen));

end;

Var

 D1, D2 : TDateTime;

Begin

 D1:=Today;

 D2:=Today-364;

 Test(D1,D2);

 D2:=Today-365;

 Test(D1,D2);

 D2:=Today-366;

 Test(D1,D2);

 D2:=Today-390;

 Test(D1,D2);

 D2:=Today-368;

 Test(D1,D2);

 D2:=Today-1000;

 Test(D1,D2);

End.

NthDayOfWeek

Synopsis: Calculate which occurrence of weekday in the month a given day represents

Declaration: `function NthDayOfWeek(const AValue: TDateTime) : Word`

Visibility: default

Description: `NthDayOfWeek` returns the occurrence of the weekday of `AValue` in the month. This is the N-th time that this weekday occurs in the month (e.g. the third saturday of the month).

See also: `EncodeDateMonthWeek` ([348](#)), `#rtl.sysutils.DayOfWeek` ([1114](#)), `DecodeDayOfWeekInMonth` ([347](#)), `EncodeDayOfWeekInMonth` ([349](#)), `TryEncodeDayOfWeekInMonth` ([402](#))

Listing: `./datutex/ex104.pp`

Program `Example104;`

{ This program demonstrates the NthDayOfWeek function }

Uses `SysUtils, DateUtils;`

Begin

`Write('Today is the ',NthDayOfWeek(Today),'-th ');`

`WriteLn(formatdateTime('dddd',Today),' of the month.');`

End.

PreviousDayOfWeek

Synopsis: Given a day of the week, return the previous day of the week.

Declaration: `function PreviousDayOfWeek(DayOfWeek: Word) : Word`

Visibility: `default`

Description: `PreviousDayOfWeek` returns the previous day of the week. If the current day is the first day of the week (1) then the last day will be returned (7).

Remark: Note that the days of the week are in ISO notation, i.e. 1-based.

See also: `Yesterday` ([420](#))

Listing: `./datutex/ex22.pp`

Program `Example22;`

{ This program demonstrates the PreviousDayOfWeek function }

Uses `SysUtils, DateUtils;`

Var

`D : Word;`

Begin

`For D:=1 to 7 do`

`WriteLn('Previous day of ',D,' is : ',PreviousDayOfWeek(D));`

End.

RecodeDate

Synopsis: Replace date part of a `TDateTime` value with another date.

Declaration: `function RecodeDate(const AValue: TDateTime;const AYear: Word;
const AMonth: Word;const ADay: Word) : TDateTime`

Visibility: default

Description: `RecodeDate` replaces the date part of the timestamp `AValue` with the date specified in `AYear`, `AMonth`, `ADay`. All other parts (the time part) of the date/time stamp are left untouched.

Errors: If one of the `AYear`, `AMonth`, `ADay` values is not within a valid range then an `EConvertError` exception is raised.

See also: `RecodeYear` (387), `RecodeMonth` (385), `RecodeDay` (382), `RecodeHour` (383), `RecodeMinute` (384), `RecodeSecond` (385), `RecodeDate` (380), `RecodeTime` (386), `RecodeDateTime` (381)

Listing: `./datutex/ex94.pp`

Program `Example94`;

{ This program demonstrates the RecodeDate function }

Uses `SysUtils`, `DateUtils`;

Const

`Fmt = 'dddd dd mmmm yyyy hh:nn:ss';`

Var

`S : AnsiString;`

Begin

`S:=FormatDateTime(Fmt,RecodeDate(Now,2001,1,1));`

`WriteLn('This moment on the first of the millenium : ',S);`

End.

RecodeDateTime

Synopsis: Replace selected parts of a `TDateTime` value with other values

Declaration: `function RecodeDateTime(const AValue: TDateTime;const AYear: Word;
const AMonth: Word;const ADay: Word;
const AHour: Word;const AMinute: Word;
const ASecond: Word;const AMilliSecond: Word)
: TDateTime`

Visibility: default

Description: `RecodeDateTime` replaces selected parts of the timestamp `AValue` with the date/time values specified in `AYear`, `AMonth`, `ADay`, `AHour`, `AMinute`, `ASecond` and `AMilliSecond`. If any of these values equals the pre-defined constant `RecodeLeaveFieldAsIs` (334), then the corresponding part of the date/time stamp is left untouched.

Errors: If one of the values `AYear`, `AMonth`, `ADay`, `AHour`, `AMinute`, `ASecond`, `AMilliSecond` is not within a valid range (`RecodeLeaveFieldAsIs` excepted) then an `EConvertError` exception is raised.

See also: `RecodeYear` (387), `RecodeMonth` (385), `RecodeDay` (382), `RecodeHour` (383), `RecodeMinute` (384), `RecodeSecond` (385), `RecodeMilliSecond` (383), `RecodeDate` (380), `RecodeTime` (386), `TryRecodeDateTime` (404)

Listing: `./datutex/ex96.pp`

Program Example96;

{ This program demonstrates the RecodeDateTime function }

Uses SysUtils, DateUtils;

Const

Fmt = 'dddd dd mmmm yyyy hh:nn:ss';

Var

S : AnsiString;

D : TDateTime;

Begin

D:=Now;

D:=RecodeDateTime(D,2000,2,RecodeLeaveFieldAsIs,0,0,0,0);

S:=FormatDateTime(Fmt,D);

WriteIn('This moment in februari 2000 : ',S);

End.

RecodeDay

Synopsis: Replace day part of a TDateTime value with another day.

Declaration: function RecodeDay(const AValue: TDateTime;const ADay: Word) : TDateTime

Visibility: default

Description: RecodeDay replaces the Day part of the timestamp AValue with ADay. All other parts of the date/time stamp are left untouched.

Errors: If the ADay value is not within a valid range (1..12) then an EConvertError exception is raised.

See also: RecodeYear ([387](#)), RecodeMonth ([385](#)), RecodeHour ([383](#)), RecodeMinute ([384](#)), RecodeSecond ([385](#)), RecodeMilliSecond ([383](#)), RecodeDate ([380](#)), RecodeTime ([386](#)), RecodeDateTime ([381](#))

Listing: ./datutex/ex89.pp

Program Example89;

{ This program demonstrates the RecodeDay function }

Uses SysUtils, DateUtils;

Const

Fmt = 'dddd dd mmmm yyyy hh:nn:ss';

Var

S : AnsiString;

Begin

S:=FormatDateTime(Fmt,RecodeDay(Now,1));

WriteIn('This moment on the first of the month : ',S);

End.

RecodeHour

Synopsis: Replace hours part of a `TDateTime` value with another hour.

Declaration: `function RecodeHour(const AValue: TDateTime; const AHour: Word) : TDateTime`

Visibility: default

Description: `RecodeHour` replaces the Hour part of the timestamp `AValue` with `AHour`. All other parts of the date/time stamp are left untouched.

Errors: If the `AHour` value is not within a valid range (0..23) then an `EConvertError` exception is raised.

See also: `RecodeYear` (387), `RecodeMonth` (385), `RecodeDay` (382), `RecodeMinute` (384), `RecodeSecond` (385), `RecodeMilliSecond` (383), `RecodeDate` (380), `RecodeTime` (386), `RecodeDateTime` (381)

Listing: ./datutex/ex90.pp

Program Example90;

{ This program demonstrates the RecodeHour function }

Uses SysUtils, DateUtils;

Const

 Fmt = 'dddd dd mmm yyyy hh:nn:ss';

Var

 S : AnsiString;

Begin

 S := **FormatDateTime**(Fmt, **RecodeHour**(**Now**, 0));

WriteLn('Today, in the first hour : ', S);

End.

RecodeMilliSecond

Synopsis: Replace milliseconds part of a `TDateTime` value with another millisecond.

Declaration: `function RecodeMilliSecond(const AValue: TDateTime; const AMilliSecond: Word) : TDateTime`

Visibility: default

Description: `RecodeMilliSecond` replaces the millisecond part of the timestamp `AValue` with `AMilliSecond`. All other parts of the date/time stamp are left untouched.

Errors: If the `AMilliSecond` value is not within a valid range (0..999) then an `EConvertError` exception is raised.

See also: `RecodeYear` (387), `RecodeMonth` (385), `RecodeDay` (382), `RecodeHour` (383), `RecodeMinute` (384), `RecodeSecond` (385), `RecodeDate` (380), `RecodeTime` (386), `RecodeDateTime` (381)

Listing: ./datutex/ex93.pp

Program Example93;

{ This program demonstrates the RecodeMilliSecond function }

Uses SysUtils, DateUtils;

Const

Fmt = 'dddd dd mmmm yyyy hh:nn:ss.zzz';

Var

S : AnsiString;

Begin

S:=**FormatDateTime**(Fmt, RecodeMilliSecond(**Now**,0));

WriteIn('This moment, milliseconds stripped : ',S);

End.

RecodeMinute

Synopsis: Replace minutse part of a TDateTime value with another minute.

Declaration: function RecodeMinute(const AValue: TDateTime;const AMinute: Word)
: TDateTime

Visibility: default

Description: RecodeMinute replaces the Minute part of the timestamp AValue with AMinute. All other parts of the date/time stamp are left untouched.

Errors: If the AMinute value is not within a valid range (0..59) then an EConvertError exception is raised.

See also: RecodeYear ([387](#)), RecodeMonth ([385](#)), RecodeDay ([382](#)), RecodeHour ([383](#)), RecodeSecond ([385](#)), RecodeMilliSecond ([383](#)), RecodeDate ([380](#)), RecodeTime ([386](#)), RecodeDateTime ([381](#))

Listing: ./datutex/ex91.pp

Program Example91;

{ This program demonstrates the RecodeMinute function }

Uses SysUtils, DateUtils;

Const

Fmt = 'dddd dd mmmm yyyy hh:nn:ss';

Var

S : AnsiString;

Begin

S:=**FormatDateTime**(Fmt, RecodeMinute(**Now**,0));

WriteIn('This moment in the first minute of the hour: ',S);

End.

RecodeMonth

Synopsis: Replace month part of a TDateTime value with another month.

Declaration: `function RecodeMonth(const AValue: TDateTime;const AMonth: Word)
: TDateTime`

Visibility: default

Description: RecodeMonth replaces the Month part of the timestamp AValue with AMonth. All other parts of the date/time stamp are left untouched.

Errors: If the AMonth value is not within a valid range (1..12) then an EConvertError exception is raised.

See also: RecodeYear (387), RecodeDay (382), RecodeHour (383), RecodeMinute (384), RecodeSecond (385), RecodeMilliSecond (383), RecodeDate (380), RecodeTime (386), RecodeDateTime (381)

Listing: ./datutex/ex88.pp

Program Example88;

{ This program demonstrates the RecodeMonth function }

Uses SysUtils, DateUtils;

Const

Fmt = 'dddd dd mmm yyyy hh:nn:ss';

Var

S : AnsiString;

Begin

S:=FormatDateTime(Fmt, RecodeMonth(Now, 5));

WriteLn('This moment in May : ', S);

End.

RecodeSecond

Synopsis: Replace seconds part of a TDateTime value with another second.

Declaration: `function RecodeSecond(const AValue: TDateTime;const ASecond: Word)
: TDateTime`

Visibility: default

Description: RecodeSecond replaces the Second part of the timestamp AValue with ASecond. All other parts of the date/time stamp are left untouched.

Errors: If the ASecond value is not within a valid range (0..59) then an EConvertError exception is raised.

See also: RecodeYear (387), RecodeMonth (385), RecodeDay (382), RecodeHour (383), RecodeMinute (384), RecodeMilliSecond (383), RecodeDate (380), RecodeTime (386), RecodeDateTime (381)

Listing: ./datutex/ex92.pp

Program Example92;

{ This program demonstrates the RecodeSecond function }

Uses SysUtils, DateUtils;

Const

Fmt = 'dddd dd mmmm yyyy hh:nn:ss';

Var

S : AnsiString;

Begin

S:=FormatDateTime(Fmt, RecodeSecond(Now, 0));

WriteLn('This moment, seconds stripped : ', S);

End.

RecodeTime

Synopsis: Replace time part of a TDateTime value with another time.

Declaration: function RecodeTime(const AValue: TDateTime; const AHour: Word;
const AMinute: Word; const ASecond: Word;
const AMilliSecond: Word) : TDateTime

Visibility: default

Description: RecodeTime replaces the time part of the timestamp AValue with the date specified in AHour, AMinute, ASecond and AMilliSecond. All other parts (the date part) of the date/time stamp are left untouched.

Errors: If one of the values AHour, AMinute, ASecond, AMilliSecond is not within a valid range then an EConvertError exception is raised.

See also: RecodeYear ([387](#)), RecodeMonth ([385](#)), RecodeDay ([382](#)), RecodeHour ([383](#)), RecodeMinute ([384](#)), RecodeSecond ([385](#)), RecodeMilliSecond ([383](#)), RecodeDate ([380](#)), RecodeDateTime ([381](#))

Listing: ./datutex/ex95.pp

Program Example95;

{ This program demonstrates the RecodeTime function }

Uses SysUtils, DateUtils;

Const

Fmt = 'dddd dd mmmm yyyy hh:nn:ss';

Var

S : AnsiString;

Begin

S:=FormatDateTime(Fmt, RecodeTime(Now, 8, 0, 0, 0));

WriteLn('Today, 8 AM : ', S);

End.

RecodeYear

Synopsis: Replace year part of a `TDateTime` value with another year.

Declaration: `function RecodeYear(const AValue: TDateTime;const AYear: Word)
: TDateTime`

Visibility: default

Description: `RecodeYear` replaces the year part of the timestamp `AValue` with `AYear`. All other parts of the date/time stamp are left untouched.

Errors: If the `AYear` value is not within a valid range (1..9999) then an `EConvertError` exception is raised.

See also: `RecodeMonth` (385), `RecodeDay` (382), `RecodeHour` (383), `RecodeMinute` (384), `RecodeSecond` (385), `RecodeMilliSecond` (383), `RecodeDate` (380), `RecodeTime` (386), `RecodeDateTime` (381)

Listing: `./datutex/ex87.pp`

Program `Example87;`

{ This program demonstrates the RecodeYear function }

Uses `SysUtils, DateUtils;`

Const

`Fmt = 'dddd dd mmm yyyy hh:nn:ss';`

Var

`S : AnsiString;`

Begin

`S:=FormatDateTime(Fmt,RecodeYear(Now,1999));`

`WriteLn('This moment in 1999 : ',S);`

End.

SameDate

Synopsis: Check whether two `TDateTime` values have the same date part.

Declaration: `function SameDate(const A: TDateTime;const B: TDateTime) : Boolean`

Visibility: default

Description: `SameDate` compares the date parts of two timestamps `A` and `B` and returns `True` if they are equal, `False` if they are not.

The function simply checks whether `CompareDate` (335) returns zero.

See also: `CompareDateTime` (336), `CompareDate` (335), `CompareTime` (337), `SameDateTime` (388), `SameTime` (389)

Listing: `./datutex/ex102.pp`

Program `Example102;`

{ This program demonstrates the SameDate function }

Uses SysUtils, DateUtils;

Const

 Fmt = 'dddd dd mmm yyyy hh:nn:ss.zzz';

Procedure Test(D1,D2 : TDateTime);

begin

Write(FormatDateTime(Fmt,D1),' is the same date as ');

WriteIn(FormatDateTime(Fmt,D2),' : ',SameDate(D1,D2));

end;

Var

 D,N : TDateTime;

Begin

 D:=Today;

 N:=Now;

 Test(D,D);

 Test(N,N);

 Test(N+1,N);

 Test(N-1,N);

 Test(N+OneSecond,N);

 Test(N-OneSecond,N);

End.

SameDateTime

Synopsis: Check whether two TDateTime values have the same date and time parts.

Declaration: function SameDateTime(const A: TDateTime;const B: TDateTime) : Boolean

Visibility: default

Description: SameDateTime compares the date/time parts of two timestamps A and B and returns True if they are equal, False if they are not.

The function simply checks whether CompareDateTime (336) returns zero.

See also: CompareDateTime (336), CompareDate (335), CompareTime (337), SameDate (387), SameTime (389)

Listing: ./datutex/ex101.pp

Program Example101;

{ This program demonstrates the SameDateTime function }

Uses SysUtils, DateUtils;

Const

 Fmt = 'dddd dd mmm yyyy hh:nn:ss.zzz';

Procedure Test(D1,D2 : TDateTime);

begin

Write(FormatDateTime(Fmt,D1),' is the same datetime as ');

WriteIn(FormatDateTime(Fmt,D2),' : ',SameDateTime(D1,D2));

```

end;

Var
  D,N : TDateTime;

Begin
  D:=Today;
  N:=Now;
  Test(D,D);
  Test(N,N);
  Test(N+1,N);
  Test(N-1,N);
  Test(N+OneSecond,N);
  Test(N-OneSecond,N);
End.

```

SameTime

Synopsis: Check whether two TDateTime values have the same time part.

Declaration: `function SameTime(const A: TDateTime;const B: TDateTime) : Boolean`

Visibility: default

Description: SameTime compares the time parts of two timestamps A and B and returns True if they are equal, False if they are not.

The function simply checks whether CompareTime (337) returns zero.

See also: CompareDateTime (336), CompareDate (335), CompareTime (337), SameDateTime (388), SameDate (387)

Listing: ./datutex/ex103.pp

Program Example102;

{ This program demonstrates the SameTime function }

Uses SysUtils, DateUtils;

Const

 Fmt = 'dddd dd mmm yyyy hh:nn:ss.zzz';

Procedure Test(D1,D2 : TDateTime);

begin

 Write(FormatDateTime(Fmt,D1), ' is the same time as ');

 WriteLn(FormatDateTime(Fmt,D2), ' : ', SameTime(D1,D2));

end;

Var

 D,N : TDateTime;

Begin

 D:=Today;

 N:=Now;

 Test(D,D);

 Test(N,N);

```

Test(N+1,N);
Test(N-1,N);
Test(N+OneSecond,N);
Test(N-OneSecond,N);
End.

```

SecondOf

Synopsis: Extract the second part from a DateTime value.

Declaration: `function SecondOf(const AValue: TDateTime) : Word`

Visibility: default

Description: `SecondOf` returns the second of the minute part of the AValue date/time indication. It is a number between 0 and 59.

For an example, see `YearOf` ([417](#))

See also: `YearOf` ([417](#)), `WeekOf` ([405](#)), `MonthOf` ([377](#)), `DayOf` ([339](#)), `HourOf` ([354](#)), `MinuteOf` ([373](#)), `MilliSecondOf` ([369](#))

SecondOfTheDay

Synopsis: Calculate the number of seconds elapsed since the start of the day

Declaration: `function SecondOfTheDay(const AValue: TDateTime) : LongWord`

Visibility: default

Description: `SecondOfTheDay` returns the number of seconds that have passed since the start of the Day (00:00:00) till the moment indicated by AValue. This is a zero-based number, i.e. 00:00:00.999 return 0.

For an example, see the `HourOfTheDay` ([354](#)) function.

See also: `SecondOfTheYear` ([392](#)), `SecondOfTheMonth` ([391](#)), `SecondOfTheWeek` ([391](#)), `SecondOfTheHour` ([390](#)), `SecondOfTheMinute` ([391](#)), `HourOfTheDay` ([354](#)), `MinuteOfTheDay` ([373](#)), `MilliSecondOfTheDay` ([369](#))

SecondOfTheHour

Synopsis: Calculate the number of seconds elapsed since the start of the hour

Declaration: `function SecondOfTheHour(const AValue: TDateTime) : Word`

Visibility: default

Description: `SecondOfTheHour` returns the number of seconds that have passed since the start of the Hour (HH:00:00) till the moment indicated by AValue. This is a zero-based number, i.e. HH:00:00.999 return 0.

For an example, see the `MinuteOfTheHour` ([374](#)) function.

See also: `SecondOfTheYear` ([392](#)), `SecondOfTheMonth` ([391](#)), `SecondOfTheWeek` ([391](#)), `SecondOfTheDay` ([390](#)), `SecondOfTheMinute` ([391](#)), `MinuteOfTheHour` ([374](#)), `MilliSecondOfTheHour` ([370](#))

SecondOfTheMinute

Synopsis: Calculate the number of seconds elapsed since the start of the minute

Declaration: `function SecondOfTheMinute(const AValue: TDateTime) : Word`

Visibility: default

Description: `SecondOfTheMinute` returns the number of seconds that have passed since the start of the minute (HH:MM:00) till the moment indicated by `AValue`. This is a zero-based number, i.e. HH:MM:00.999 return 0.

See also: `SecondOfTheYear` ([392](#)), `SecondOfTheMonth` ([391](#)), `SecondOfTheWeek` ([391](#)), `SecondOfTheDay` ([390](#)), `SecondOfTheHour` ([390](#)), `MilliSecondOfTheMinute` ([370](#))

Listing: `./datutex/ex45.pp`

Program `Example45;`

`{ This program demonstrates the SecondOfTheMinute function }`

Uses `SysUtils, DateUtils;`

Var

`N : TDateTime;`

Begin

`N:=Now;`

`WriteLn('Second of the Minute : ',SecondOfTheMinute(N));`

`WriteLn('MilliSecond of the Minute : ',`
`MilliSecondOfTheMinute(N));`

End.

SecondOfTheMonth

Synopsis: Calculate number of seconds elapsed since the start of the month.

Declaration: `function SecondOfTheMonth(const AValue: TDateTime) : LongWord`

Visibility: default

Description: `SecondOfTheMonth` returns the number of seconds that have passed since the start of the month (00:00:00) till the moment indicated by `AValue`. This is a zero-based number, i.e. 00:00:00.999 on the first day of the month will return 0.

For an example, see the `WeekOfTheMonth` ([405](#)) function.

See also: `WeekOfTheMonth` ([405](#)), `DayOfTheMonth` ([340](#)), `HourOfTheMonth` ([355](#)), `MinuteOfTheMonth` ([374](#)), `MilliSecondOfTheMonth` ([370](#))

SecondOfTheWeek

Synopsis: Calculate the number of seconds elapsed since the start of the week

Declaration: `function SecondOfTheWeek(const AValue: TDateTime) : LongWord`

Visibility: default

Description: `SecondOfTheWeek` returns the number of seconds that have passed since the start of the week (00:00:00) till the moment indicated by `AValue`. This is a zero-based number, i.e. 00:00:00.999 on the first day of the week will return 0.

For an example, see the `DayOfTheWeek` (340) function.

See also: `SecondOfTheYear` (392), `SecondOfTheMonth` (391), `SecondOfTheDay` (390), `SecondOfTheHour` (390), `SecondOfTheMinute` (391), `DayOfTheWeek` (340), `HourOfTheWeek` (355), `MinuteOfTheWeek` (375), `MilliSecondOfTheWeek` (371)

SecondOfTheYear

Synopsis: Calculate the number of seconds elapsed since the start of the year.

Declaration: `function SecondOfTheYear(const AValue: TDateTime) : LongWord`

Visibility: default

Description: `SecondOfTheYear` returns the number of seconds that have passed since the start of the year (January 1, 00:00:00) till the moment indicated by `AValue`. This is a zero-based number, i.e. January 1 00:00:00.999 will return 0.

For an example, see the `WeekOfTheYear` (406) function.

See also: `WeekOfTheYear` (406), `DayOfTheYear` (341), `HourOfTheYear` (356), `MinuteOfTheYear` (375), `SecondOfTheYear` (392), `MilliSecondOfTheYear` (371)

SecondsBetween

Synopsis: Calculate the number of whole seconds between two `DateTime` values.

Declaration: `function SecondsBetween(const ANow: TDateTime;const AThen: TDateTime) : Int64`

Visibility: default

Description: `SecondsBetween` returns the number of whole seconds between `ANow` and `AThen`. This means the fractional part of a second (milliseconds etc.) is dropped.

See also: `YearsBetween` (418), `MonthsBetween` (378), `WeeksBetween` (407), `DaysBetween` (341), `HoursBetween` (356), `MinutesBetween` (375), `MilliSecondsBetween` (372)

Listing: `./datutex/ex61.pp`

Program `Example61`;

{ This program demonstrates the SecondsBetween function }

Uses `SysUtils`, `DateUtils`;

Procedure `Test(ANow, AThen : TDateTime);`

begin

`Write('Number of seconds between ');`
`Write(TimeToStr(AThen), ' and ', TimeToStr(ANow));`
`WriteLn(' : ', SecondsBetween(ANow, AThen));`
end;

Var

D1,D2 : TDateTime;

Begin

```
D1:=Now;
D2:=D1-(999*OneMilliSecond);
Test(D1,D2);
D2:=D1-(1001*OneMilliSecond);
Test(D1,D2);
D2:=D1-(2001*OneMilliSecond);
Test(D1,D2);
D2:=D1-(5001*OneMilliSecond);
Test(D1,D2);
D2:=D1-(5.4*OneSecond);
Test(D1,D2);
D2:=D1-(2.5*OneSecond);
Test(D1,D2);
```

End.

SecondSpan

Synopsis: Calculate the approximate number of seconds between two DateTime values.

Declaration: `function SecondSpan(const ANow: TDateTime;const AThen: TDateTime)
: Double`

Visibility: default

Description: SecondSpan returns the number of seconds between ANow and AThen, including any fractional parts of a second.

See also: YearSpan ([419](#)), MonthSpan ([379](#)), WeekSpan ([409](#)), DaySpan ([344](#)), HourSpan ([357](#)), MinuteSpan ([376](#)), MilliSecondSpan ([372](#)), SecondsBetween ([392](#))

Listing: ./datutex/ex69.pp

Program Example69;

{ This program demonstrates the SecondSpan function }

Uses SysUtils, DateUtils;

Procedure Test(ANow, AThen : TDateTime);

begin

```
Write('Number of seconds between ');
Write(TimeToStr(AThen), ' and ', TimeToStr(ANow));
WriteLn(' : ', SecondSpan(ANow, AThen));
end;
```

Var

D1,D2 : TDateTime;

Begin

```
D1:=Now;
D2:=D1-(999*OneMilliSecond);
Test(D1,D2);
D2:=D1-(1001*OneMilliSecond);
```

```

Test(D1,D2);
D2:=D1-(2001*OneMilliSecond);
Test(D1,D2);
D2:=D1-(5001*OneMilliSecond);
Test(D1,D2);
D2:=D1-(5.4*OneSecond);
Test(D1,D2);
D2:=D1-(2.5*OneSecond);
Test(D1,D2);
End.

```

StartOfDay

Synopsis: Return the start of a day as a DateTime value, given a day indication

Declaration:

```

function StartOfDay(const AYear: Word;const AMonth: Word;
                    const ADay: Word) : TDateTime; Overload
function StartOfDay(const AYear: Word;const ADayOfYear: Word)
                    : TDateTime; Overload

```

Visibility: default

Description: StartOfDay returns a TDateTime value with the date/time indication of the start (0:0:0.000) of the day given by AYear, AMonth, ADay.

The day may also be indicated with a AYear, ADayOfYear pair.

See also: StartOfDay (396), StartOfTheWeek (397), StartOfAWeek (395), StartOfAMonth (395), StartOfTheMonth (397), EndOfTheWeek (353), EndOfAWeek (351), EndOfTheYear (354), EndOfAYear (352), EndOfTheMonth (353), EndOfAMonth (350), EndOfTheDay (352), EndOfDay (350)

Listing: ./datutex/ex38.pp

Program Example38;

{ This program demonstrates the StartOfDay function }

Uses SysUtils, DateUtils;

Const

 Fmt = ' "Start of the day : " dd mmm yyyy hh:nn:ss ';

Var

 Y,M,D : Word;

Begin

 Y:=YearOf(Today);

 M:=MonthOf(Today);

 D:=DayOf(Today);

 WriteLn(FormatDateTime(Fmt, StartOfDay(Y,M,D)));

 DecodeDateDay(Today,Y,D);

 WriteLn(FormatDateTime(Fmt, StartOfDay(Y,D)));

End.

StartOfAMonth

Synopsis: Return first date of month, given a year/month pair.

Declaration: `function StartOfAMonth(const AYear: Word;const AMonth: Word) : TDateTime`

Visibility: default

Description: `StartOfAMonth` returns a `TDateTime` value with the date of the first day of the month indicated by the `AYear`, `AMonth` pair.

See also: `StartOfTheMonth` (397), `EndOfTheMonth` (353), `EndOfAMonth` (350), `EndOfTheYear` (354), `EndOfAYear` (352), `StartOfAWeek` (395), `StartOfTheWeek` (397)

Listing: `./datutex/ex30.pp`

Program Example30;

{ This program demonstrates the StartOfAMonth function }

Uses SysUtils, DateUtils;

Const

 Fmt = ' "First day of this month : " dd mmm yyyy ';

Var

 Y,M : Word;

Begin

 Y:=YearOf(Today);

 M:=MonthOf(Today);

WriteLn (**FormatDateTime** (Fmt, StartOfAMonth (Y,M)));

End.

StartOfAWeek

Synopsis: Return a day of the week, given a year, week and day in the week.

Declaration: `function StartOfAWeek(const AYear: Word;const AWeekOfYear: Word;
 const ADayOfWeek: Word) : TDateTime`
`function StartOfAWeek(const AYear: Word;const AWeekOfYear: Word)
 : TDateTime`

Visibility: default

Description: `StartOfAWeek` returns a `TDateTime` value with the date of the indicated day of the week indicated by the `AYear`, `AWeek`, `ADayOfWeek` values.

The default value for `ADayOfWeek` is 1.

See also: `StartOfTheWeek` (397), `EndOfTheWeek` (353), `EndOfAWeek` (351), `StartOfAMonth` (395), `EndOfTheYear` (354), `EndOfAYear` (352), `EndOfTheMonth` (353), `EndOfAMonth` (350)

Listing: `./datutex/ex34.pp`

Program Example34;

{ This program demonstrates the StartOfAWeek function }

Uses SysUtils, DateUtils;

Const

```
Fmt = '"First day of this week : "dd mmm yyyy hh:nn:ss';
Fmt2 = '"Second day of this week : "dd mmm yyyy hh:nn:ss';
```

Var

```
Y,W : Word;
```

Begin

```
Y:=YearOf(Today);
W:=WeekOf(Today);
WriteIn(FormatDateTime(Fmt, StartOfAWeek(Y,W)));
WriteIn(FormatDateTime(Fmt2, StartOfAWeek(Y,W,2)));
```

End.

StartOfAYear

Synopsis: Return the first day of a given year.

Declaration: `function StartOfAYear(const AYear: Word) : TDateTime`

Visibility: default

Description: `StartOfAYear` returns a `TDateTime` value with the date of the first day of the year `AYear` (January 1).

See also: `StartOfTheYear` ([398](#)), `EndOfTheYear` ([354](#)), `EndOfAYear` ([352](#)), `EndOfTheMonth` ([353](#)), `EndOfA-Month` ([350](#)), `StartOfAWeek` ([395](#)), `StartOfTheWeek` ([397](#))

Listing: `./datutex/ex26.pp`

Program Example26;

```
{ This program demonstrates the StartOfAYear function }
```

Uses SysUtils, DateUtils;

Const

```
Fmt = '"First day of this year : "dd mmm yyyy';
```

Begin

```
WriteIn(FormatDateTime(Fmt, StartOfAYear(YearOf(Today))));
```

End.

StartOfTheDay

Synopsis: Calculate the start of the day as a `DateTime` value, given a moment in the day.

Declaration: `function StartOfTheDay(const AValue: TDateTime) : TDateTime`

Visibility: default

Description: `StartOfTheDay` extracts the date part of `AValue` and returns a `TDateTime` value with the date/time indication of the start (0:0:0.000) of this day.

See also: [StartOfADay \(394\)](#), [StartOfTheWeek \(397\)](#), [StartOfAWeek \(395\)](#), [StartOfAMonth \(395\)](#), [StartOfTheMonth \(397\)](#), [EndOfTheWeek \(353\)](#), [EndOfAWeek \(351\)](#), [EndOfTheYear \(354\)](#), [EndOfAYear \(352\)](#), [EndOfTheMonth \(353\)](#), [EndOfAMonth \(350\)](#), [EndOftheDay \(352\)](#), [EndOfADay \(350\)](#)

Listing: ./datutex/ex36.pp

Program Example36;

{ This program demonstrates the StartOfTheDay function }

Uses SysUtils , DateUtils ;

Const

Fmt = ' "Start of the day : " dd mmm yyyy hh:nn:ss ';

Begin

WriteIn (FormatDateTime (Fmt, StartOfTheDay (Today)));

End.

StartOfTheMonth

Synopsis: Calculate the first day of the month, given a date in that month.

Declaration: function StartOfTheMonth(const AValue: TDateTime) : TDateTime

Visibility: default

Description: StartOfTheMonth extracts the year and month parts of AValue and returns a TDateTime value with the date of the first day of that year and month as the StartOfAMonth ([395](#)) function.

See also: [StartOfAMonth \(395\)](#), [EndOfTheYear \(354\)](#), [EndOfAYear \(352\)](#), [EndOfTheMonth \(353\)](#), [EndOfAMonth \(350\)](#), [StartOfAWeek \(395\)](#), [StartOfTheWeek \(397\)](#)

Listing: ./datutex/ex28.pp

Program Example28;

{ This program demonstrates the StartOfTheMonth function }

Uses SysUtils , DateUtils ;

Const

Fmt = ' "First day of this month : " dd mmm yyyy ';

Begin

WriteIn (FormatDateTime (Fmt, StartOfTheMonth (Today)));

End.

StartOfTheWeek

Synopsis: Return the first day of the week, given a date.

Declaration: function StartOfTheWeek(const AValue: TDateTime) : TDateTime

Visibility: default

Description: `StartOfTheWeek` extracts the year and week parts of `AValue` and returns a `TDateTime` value with the date of the first day of that week as the `StartOfAWeek` (395) function.

See also: `StartOfAWeek` (395), `EndOfTheWeek` (353), `EndOfAWeek` (351), `StartOfAMonth` (395), `EndOfTheYear` (354), `EndOfAYear` (352), `EndOfTheMonth` (353), `EndOfAMonth` (350)

Listing: `./datutex/ex32.pp`

Program `Example32;`

`{ This program demonstrates the StartOfTheWeek function }`

Uses `SysUtils, DateUtils;`

Const

`Fmt = ' "First day of this week : " dd mmm yyyy ';`

Begin

`WriteLn (FormatDateTime (Fmt, StartOfTheWeek (Today)));`

End.

StartOfTheYear

Synopsis: Return the first day of the year, given a date in this year.

Declaration: `function StartOfTheYear(const AValue: TDateTime) : TDateTime`

Visibility: `default`

Description: `StartOfTheYear` extracts the year part of `AValue` and returns a `TDateTime` value with the date of the first day of that year (January 1), as the `StartOfAYear` (396) function.

See also: `StartOfAYear` (396), `EndOfTheYear` (354), `EndOfAYear` (352)

Listing: `./datutex/ex24.pp`

Program `Example24;`

`{ This program demonstrates the StartOfTheYear function }`

Uses `SysUtils, DateUtils;`

Const

`Fmt = ' "First day of this year : " dd mmm yyyy ';`

Begin

`WriteLn (FormatDateTime (Fmt, StartOfTheYear (Today)));`

End.

TimeOf

Synopsis: Extract the time part from a `DateTime` indication.

Declaration: `function TimeOf(const AValue: TDateTime) : TDateTime`

Visibility: `default`

Description: TimeOf extracts the time part from AValue and returns the result.

Since the TDateTime is actually a double with the time part encoded in the fractional part, this operation corresponds to a call to Frac.

See also: DateOf (338), YearOf (417), MonthOf (377), DayOf (339), HourOf (354), MinuteOf (373), SecondOf (390), MilliSecondOf (369)

Listing: ./datutex/ex2.pp

Program Example2;

{ This program demonstrates the TimeOf function }

Uses SysUtils , DateUtils ;

Begin

WriteIn ('Time is : ' , TimeToStr (TimeOf (Now)));
End.

Today

Synopsis: Return the current date

Declaration: function Today : TDateTime

Visibility: default

Description: Today is an alias for the Date (1110) function in the sysutils (1071) unit.

For an example, see Yesterday (420)

See also: #rtl.sysutils.Date (1110), Yesterday (420), Tomorrow (399)

Tomorrow

Synopsis: Return the next day

Declaration: function Tomorrow : TDateTime

Visibility: default

Description: Tomorrow returns tomorrow's date. Tomorrow is determined from the system clock, i.e. it is Today (399) +1.

See also: Today (399), Yesterday (420)

Listing: ./datutex/ex19.pp

Program Example19;

{ This program demonstrates the Tomorrow function }

Uses SysUtils , DateUtils ;

Begin

WriteIn (FormatDateTime (' "Today is " dd mmm yyyy ' , Today));
WriteIn (FormatDateTime (' "Tomorrow will be " dd mmm yyyy ' , Tomorrow));
End.

TryEncodeDateDay

Synopsis: Encode a year and day of year to a TDateTime value

Declaration: `function TryEncodeDateDay(const AYear: Word;const ADayOfYear: Word;
var AValue: TDateTime) : Boolean`

Visibility: default

Description: TryEncodeDateDay encodes the values AYear and ADayOfYear to a date value and returns this value in AValue.

If the encoding was successful, True is returned. False is returned if any of the arguments is not valid.

See also: EncodeDateDay (348), EncodeDateTime (348), EncodeDateMonthWeek (348), EncodeDateWeek (349), TryEncodeDateTime (401), TryEncodeDateMonthWeek (400), TryEncodeDateWeek (402)

Listing: ./datutex/ex84.pp

Program Example84;

{ This program demonstrates the TryEncodeDateDay function }

Uses SysUtils, DateUtils;

Var

Y, DoY : Word;
TS : TDateTime;

Begin

DecodeDateDay(**Now**, Y, DoY);
If TryEncodeDateDay(Y, DoY, TS) **then**
 WriteLn('Today is : ', **DateToStr**(TS))
else
 WriteLn('Wrong year/day of year indication ');

End.

TryEncodeDateMonthWeek

Synopsis: Encode a year, month, week of month and day of week to a TDateTime value

Declaration: `function TryEncodeDateMonthWeek(const AYear: Word;const AMonth: Word;
const AWeekOfMonth: Word;
const ADayOfWeek: Word;
var AValue: TDateTime) : Boolean`

Visibility: default

Description: TryEncodeDateMonthWeek encodes the values AYear, AMonth, AWeekOfMonth, ADayOfWeek, to a date value and returns this value in AValue.

If the encoding was successful, True is returned, False if any of the arguments is not valid.

See also: DecodeDateMonthWeek (345), EncodeDateTime (348), EncodeDateWeek (349), EncodeDateDay (348), EncodeDateMonthWeek (348), TryEncodeDateTime (401), TryEncodeDateWeek (402), TryEncodeDateDay (400), NthDayOfWeek (379)

Listing: ./datutex/ex86.pp

Program Example86;

{ This program demonstrates the TryEncodeDateMonthWeek function }

Uses SysUtils, DateUtils;

Var

Y, M, WoM, Dow : Word;
TS : TDateTime;

Begin

DecodeDateMonthWeek(**Now**, Y, M, WoM, DoW);
If TryEncodeDateMonthWeek(Y, M, WoM, Dow, TS) **then**
 WriteLn('Today is : ', **DateToStr**(TS))
else
 WriteLn('Invalid year/month/week/dow indication ');

End.

TryEncodeDateTime

Synopsis: Encode a Year, Month, Day, Hour, minute, seconds, milliseconds tuple to a TDateTime value

Declaration: function TryEncodeDateTime(const AYear: Word; const AMonth: Word;
 const ADay: Word; const AHour: Word;
 const AMinute: Word; const ASecond: Word;
 const AMilliSecond: Word;
 var AValue: TDateTime) : Boolean

Visibility: default

Description: EncodeDateTime encodes the values AYear, AMonth, ADay, AHour, AMinute, ASecond and AMilliSecond to a date/time value and returns this value in AValue.

If the date was encoded successfully, True is returned, False is returned if one of the arguments is not valid.

See also: EncodeDateTime ([348](#)), EncodeDateMonthWeek ([348](#)), EncodeDateWeek ([349](#)), EncodeDateDay ([348](#)), TryEncodeDateDay ([400](#)), TryEncodeDateWeek ([402](#)), TryEncodeDateMonthWeek ([400](#))

Listing: ./datutex/ex80.pp

Program Example79;

{ This program demonstrates the TryEncodeDateTime function }

Uses SysUtils, DateUtils;

Var

Y, Mo, D, H, Mi, S, MS : Word;
TS : TDateTime;

Begin

DecodeDateTime(**Now**, Y, Mo, D, H, Mi, S, MS);
If TryEncodeDateTime(Y, Mo, D, H, Mi, S, MS, TS) **then**
 WriteLn('Now is : ', **DateTimeToStr**(TS))
else
 WriteLn('Wrong date/time indication ');

End.

TryEncodeDateWeek

Synopsis: Encode a year, week and day of week triplet to a TDateTime value

Declaration:

```
function TryEncodeDateWeek(const AYear: Word;const AWeekOfYear: Word;
                           var AValue: TDateTime;const ADayOfWeek: Word)
                           : Boolean
function TryEncodeDateWeek(const AYear: Word;const AWeekOfYear: Word;
                           var AValue: TDateTime) : Boolean
```

Visibility: default

Description: TryEncodeDateWeek encodes the values AYear, AWeekOfYear and ADayOfWeek to a date value and returns this value in AValue.

If the encoding was succesful, True is returned. False is returned if any of the arguments is not valid.

See also: EncodeDateMonthWeek (348), EncodeDateWeek (349), EncodeDateTime (348), EncodeDateDay (348), TryEncodeDateTime (401), TryEncodeDateMonthWeek (400), TryEncodeDateDay (400)

Listing: ./datutex/ex82.pp

Program Example82;

{ This program demonstrates the TryEncodeDateWeek function }

Uses SysUtils , DateUtils ;

Var

Y,W,Dow : Word;
TS : TDateTime;

Begin

DecodeDateWeek(**Now**,Y,W,Dow);
If TryEncodeDateWeek(Y,W,TS,Dow) **then**
 WriteLn('Today is : ',DateToStr(TS))
else
 WriteLn('Invalid date/week indication');

End.

TryEncodeDayOfWeekInMonth

Synopsis: Encode a year, month, week, day of week triplet to a TDateTime value

Declaration:

```
function TryEncodeDayOfWeekInMonth(const AYear: Word;const AMonth: Word;
                                   const ANthDayOfWeek: Word;
                                   const ADayOfWeek: Word;
                                   var AValue: TDateTime) : Boolean
```

Visibility: default

Description: `EncodeDayOfWeekInMonth` encodes `AYear`, `AMonth`, `ADayOfWeek` and `ANthDayOfWeek` to a valid date stamp and returns the result in `AValue`.

`ANthDayOfWeek` is the N-th time that this weekday occurs in the month, e.g. the third saturday of the month.

The function returns `True` if the encoding was succesful, `False` if any of the values is not in range.

See also: `NthDayOfWeek` (379), `EncodeDateMonthWeek` (348), `#rtl.sysutils.DayOfWeek` (1114), `DecodeDayOfWeekInMonth` (347), `EncodeDayOfWeekInMonth` (349)

Listing: `./datutex/ex106.pp`

Program `Example105`;

{ This program demonstrates the `DecodeDayOfWeekInMonth` function }

Uses `SysUtils` , `DateUtils` ;

Var

`Y,M,NDoW,DoW` : `Word`;

`D` : `TDateTime`;

Begin

`DecodeDayOfWeekInMonth (Date , Y,M,NDoW,DoW)` ;

If `TryEncodeDayOfWeekInMonth (Y,M,NDoW,DoW,D)` **then**

begin

Write (`DateToStr(D)` , ' is the ' , `NDoW` , '—th ') ;

WriteIn (`formatdateTime ('dddd',D)` , ' of the month. ') ;

end

else

WriteIn ('Invalid year/month/NthDayOfWeek combination ') ;

End.

TryJulianDateToDateTime

Synopsis: Convert a Julian date representation to a `TDateTime` value.

Declaration: `function TryJulianDateToDateTime(const AValue: Double;`
`var ADateTime: TDateTime) : Boolean`

Visibility: `default`

Description: Not yet implemented.

Errors: Currently, trying to use this function will raise an exception.

See also: `DateTimeToJulianDate` (339), `JulianDateToDateTime` (369), `DateTimeToModifiedJulianDate` (339), `TryModifiedJulianDateToDateTime` (403)

TryModifiedJulianDateToDateTime

Synopsis: Convert a modified Julian date representation to a `TDateTime` value.

Declaration: `function TryModifiedJulianDateToDateTime(const AValue: Double;`
`var ADateTime: TDateTime)`
`: Boolean`

Visibility: `default`

Description: Not yet implemented.

Errors: Currently, trying to use this function will raise an exception.

See also: [DateTimeToJulianDate \(339\)](#), [JulianDateToDateTime \(369\)](#), [TryJulianDateToDateTime \(403\)](#), [DateTimeToModifiedJulianDate \(339\)](#), [ModifiedJulianDateToDateTime \(377\)](#)

TryRecodeDateTime

Synopsis: Replace selected parts of a TDateTime value with other values

Declaration:

```
function TryRecodeDateTime(const AValue: TDateTime;const AYear: Word;
                           const AMonth: Word;const ADay: Word;
                           const AHour: Word;const AMinute: Word;
                           const ASecond: Word;const AMilliSecond: Word;
                           var AResult: TDateTime) : Boolean
```

Visibility: default

Description: TryRecodeDateTime replaces selected parts of the timestamp AValue with the date/time values specified in AYear, AMonth, ADay, AHour, AMinute, ASecond and AMilliSecond. If any of these values equals the pre-defined constant RecodeLeaveFieldAsIs ([334](#)), then the corresponding part of the date/time stamp is left untouched.

The resulting date/time is returned in AValue.

The function returns True if the encoding was succesful. It returns False if one of the values AYear, AMonth, ADay, AHour, AMinute, ASecondAMilliSecond is not within a valid range.

See also: [RecodeYear \(387\)](#), [RecodeMonth \(385\)](#), [RecodeDay \(382\)](#), [RecodeHour \(383\)](#), [RecodeMinute \(384\)](#), [RecodeSecond \(385\)](#), [RecodeMilliSecond \(383\)](#), [RecodeDate \(380\)](#), [RecodeTime \(386\)](#), [RecodeDateTime \(381\)](#)

Listing: ./datutex/ex97.pp

Program Example97;

{ This program demonstrates the TryRecodeDateTime function }

Uses SysUtils , DateUtils ;

Const

Fmt = 'dddd dd mmmm yyyy hh:nn:ss' ;

Var

S : AnsiString ;

D : TDateTime ;

Begin

If TryRecodeDateTime(**Now**,2000,2,RecodeLeaveFieldAsIs,0,0,0,0,D) **then**
begin
 S:=**FormatDateTime**(Fmt,D);
WriteLn('This moment in february 2000 : ',S);
end

else

WriteLn('This moment did/does not exist in february 2000');

End.

UnixToDateTime

Synopsis: Convert Unix epoch time to a TDateTime value

Declaration: `function UnixToDateTime(const AValue: Int64) : TDateTime`

Visibility: default

Description: Not yet implemented.

Errors: Currently, trying to use this function will raise an exception.

See also: [DateTimeToUnix \(339\)](#)

WeekOf

Synopsis: Extract week (of the year) from a given date.

Declaration: `function WeekOf(const AValue: TDateTime) : Word`

Visibility: default

Description: `WeekOf` returns the week-of-the-year part of the AValue date/time indication. It is a number between 1 and 53.

For an example, see [YearOf \(417\)](#)

See also: [YearOf \(417\)](#), [DayOf \(339\)](#), [MonthOf \(377\)](#), [HourOf \(354\)](#), [MinuteOf \(373\)](#), [SecondOf \(390\)](#), [MilliSecondOf \(369\)](#)

WeekOfTheMonth

Synopsis: Extract the week of the month (and optionally month and year) from a DateTime value

Declaration: `function WeekOfTheMonth(const AValue: TDateTime) : Word; Overload`
`function WeekOfTheMonth(const AValue: TDateTime; var AYear: Word;`
`var AMonth: Word) : Word; Overload`

Visibility: default

Description: `WeekOfTheMonth` extracts the week of the month from AValue and returns it, and optionally returns the year and month as well (in AYear, AMonth respectively).

Remark: Note that weeks are numbered from 1 using the ISO 8601 standard, and the day of the week as well. This means that the year and month may not be the same as the year part of the date, since the week may start in the previous year as the first week of the year is the week with at least 4 days in it.

See also: [WeekOfTheYear \(406\)](#), [DayOfTheMonth \(340\)](#), [HourOfTheMonth \(355\)](#), [MinuteOfTheMonth \(374\)](#), [SecondOfTheMonth \(391\)](#), [MilliSecondOfTheMonth \(370\)](#)

Listing: `./datutex/ex41.pp`

Program `Example41;`

`{ This program demonstrates the WeekOfTheMonth function }`

Uses `SysUtils, DateUtils;`

Var

`N : TDateTime;`

```

Begin
  N:=Now;
  WriteLn('Week of the Month      : ',WeekOfTheMonth(N));
  WriteLn('Day of the Month       : ',DayOfTheMonth(N));
  WriteLn('Hour of the Month      : ',HourOfTheMonth(N));
  WriteLn('Minute of the Month    : ',MinuteOfTheMonth(N));
  WriteLn('Second of the Month    : ',SecondOfTheMonth(N));
  WriteLn('MilliSecond of the Month : ',
    MilliSecondOfTheMonth(N));
End.

```

WeekOfTheYear

Synopsis: Extract the week of the year (and optionally year) of a DateTime indication.

Declaration: `function WeekOfTheYear(const AValue: TDateTime) : Word; Overload`
`function WeekOfTheYear(const AValue: TDateTime;var AYear: Word) : Word`
`; Overload`

Visibility: default

Description: `WeekOfTheYear` extracts the week of the year from `AValue` and returns it, and optionally returns the year as well. It returns the same value as `WeekOf` ([405](#)).

Remark: Note that weeks are numbered from 1 using the ISO 8601 standard, and the day of the week as well. This means that the year may not be the same as the year part of the date, since the week may start in the previous year as the first week of the year is the week with at least 4 days in it.

See also: `WeekOf` ([405](#)), `MonthOfTheYear` ([377](#)), `DayOfTheYear` ([341](#)), `HourOfTheYear` ([356](#)), `MinuteOfTheYear` ([375](#)), `SecondOfTheYear` ([392](#)), `MilliSecondOfTheYear` ([371](#))

Listing: `./datutex/ex40.pp`

Program Example40;

{ This program demonstrates the WeekOfTheYear function }

Uses SysUtils, DateUtils;

Var

N : TDateTime;

Begin

```

  N:=Now;
  WriteLn('Month of the year      : ',MonthOfTheYear(N));
  WriteLn('Week of the year       : ',WeekOfTheYear(N));
  WriteLn('Day of the year        : ',DayOfTheYear(N));
  WriteLn('Hour of the year        : ',HourOfTheYear(N));
  WriteLn('Minute of the year     : ',MinuteOfTheYear(N));
  WriteLn('Second of the year     : ',SecondOfTheYear(N));
  WriteLn('MilliSecond of the year : ',
    MilliSecondOfTheYear(N));

```

End.

WeeksBetween

Synopsis: Calculate the number of whole weeks between two DateTime values

Declaration: `function WeeksBetween(const ANow: TDateTime; const AThen: TDateTime)
: Integer`

Visibility: default

Description: `WeeksBetween` returns the number of whole weeks between `ANow` and `AThen`. This means the fractional part of a Week is dropped.

See also: `YearsBetween` (418), `MonthsBetween` (378), `DaysBetween` (341), `HoursBetween` (356), `MinutesBetween` (375), `SecondsBetween` (392), `MillisecondsBetween` (372)

Listing: `./datutex/ex57.pp`

Program `Example57`;

{ This program demonstrates the WeeksBetween function }

Uses `SysUtils`, `DateUtils`;

Procedure `Test(ANow, AThen : TDateTime)`;

begin

Write('Number of weeks between ');

Write(**DateToStr**(AThen), ' and ', **DateToStr**(ANow));

WriteLn(' : ', `WeeksBetween`(ANow, AThen));

end;

Var

 D1, D2 : TDateTime;

Begin

 D1 := Today;

 D2 := Today - 7;

 Test(D1, D2);

 D2 := Today - 8;

 Test(D1, D2);

 D2 := Today - 14;

 Test(D1, D2);

 D2 := Today - 35;

 Test(D1, D2);

 D2 := Today - 36;

 Test(D1, D2);

 D2 := Today - 17;

 Test(D1, D2);

End.

WeeksInAYear

Synopsis: Return the number of weeks in a given year

Declaration: `function WeeksInAYear(const AYear: Word) : Word`

Visibility: default

Description: `WeeksInAYear` returns the number of weeks in the year `AYear`. The return value is either 52 or 53.

Remark: The first week of the year is determined according to the ISO 8601 standard: It is the first week that has at least 4 days in it, i.e. it includes a thursday.

See also: `WeeksInYear` ([408](#)), `DaysInYear` ([343](#)), `DaysInAYear` ([342](#)), `DaysInMonth` ([343](#)), `DaysInAMonth` ([342](#))

Listing: `./datutex/ex13.pp`

Program `Example13;`

{ This program demonstrates the WeeksInAYear function }

Uses `SysUtils, DateUtils;`

Var

`Y : Word;`

Begin

`For Y:=1992 to 2010 do`

`WriteLn(Y, ' has ', WeeksInAYear(Y), ' weeks. ');`

End.

WeeksInYear

Synopsis: return the number of weeks in the year, given a date

Declaration: `function WeeksInYear(const AValue: TDateTime) : Word`

Visibility: `default`

Description: `WeeksInYear` returns the number of weeks in the year part of `AValue`. The return value is either 52 or 53.

Remark: The first week of the year is determined according to the ISO 8601 standard: It is the first week that has at least 4 days in it, i.e. it includes a thursday.

See also: `WeeksInAYear` ([407](#)), `DaysInYear` ([343](#)), `DaysInAYear` ([342](#)), `DaysInMonth` ([343](#)), `DaysInAMonth` ([342](#))

Listing: `./datutex/ex12.pp`

Program `Example12;`

{ This program demonstrates the WeeksInYear function }

Uses `SysUtils, DateUtils;`

Var

`Y : Word;`

Begin

`For Y:=1992 to 2010 do`

`WriteLn(Y, ' has ', WeeksInYear(EncodeDate(Y,2,1)), ' weeks. ');`

End.

WeekSpan

Synopsis: Calculate the approximate number of weeks between two DateTime values.

Declaration: `function WeekSpan(const ANow: TDateTime;const AThen: TDateTime) : Double`

Visibility: default

Description: `WeekSpan` returns the number of weeks between `ANow` and `AThen`, including any fractional parts of a week.

See also: `YearSpan` ([419](#)), `MonthSpan` ([379](#)), `DaySpan` ([344](#)), `HourSpan` ([357](#)), `MinuteSpan` ([376](#)), `SecondSpan` ([393](#)), `MilliSecondSpan` ([372](#)), `WeeksBetween` ([407](#))

Listing: `./datutex/ex65.pp`

Program `Example57;`

{ This program demonstrates the WeekSpan function }

Uses `SysUtils, DateUtils;`

Procedure `Test(ANow, AThen : TDateTime);`

begin

`Write('Number of weeks between ');`

`Write(DateToStr(AThen), ' and ', DateToStr(ANow));`

`WriteLn(' : ', WeekSpan(ANow, AThen));`

end;

Var

`D1, D2 : TDateTime;`

Begin

`D1:=Today;`

`D2:=Today-7;`

`Test(D1,D2);`

`D2:=Today-8;`

`Test(D1,D2);`

`D2:=Today-14;`

`Test(D1,D2);`

`D2:=Today-35;`

`Test(D1,D2);`

`D2:=Today-36;`

`Test(D1,D2);`

`D2:=Today-17;`

`Test(D1,D2);`

End.

WithinPastDays

Synopsis: Check whether two datetimes are only a number of days apart

Declaration: `function WithinPastDays(const ANow: TDateTime;const AThen: TDateTime;
const ADays: Integer) : Boolean`

Visibility: default

Description: `WithinPastDays` compares the timestamps `ANow` and `AThen` and returns `True` if the difference between them is at most `ADays` days apart, or `False` if they are further apart.

Remark: Since this function uses the `DaysBetween` (341) function to calculate the difference in days, this means that fractional days do not count, and the fractional part is simply dropped, so for two dates actually 2 and a half days apart, the result will also be `True`

See also: `WithinPastYears` (416), `WithinPastMonths` (413), `WithinPastWeeks` (415), `WithinPastHours` (410), `WithinPastMinutes` (412), `WithinPastSeconds` (414), `WithinPastMilliseconds` (411)

Listing: `./datutex/ex50.pp`

Program `Example50`;

{ This program demonstrates the WithinPastDays function }

Uses `SysUtils`, `DateUtils`;

Procedure `Test(ANow, AThen : TDateTime; ADays : Integer);`

begin

`Write(DateTimeToStr(AThen), ' and ', DateTimeToStr(ANow));`

`Write(' are within ', ADays, ' days: ');`

`WriteLn(WithinPastDays(ANow, AThen, ADays));`

end;

Var

`D1, D2 : TDateTime;`

Begin

`D1:=Now;`

`D2:=Today-23/24;`

`Test(D1,D2,1);`

`D2:=Today-1;`

`Test(D1,D2,1);`

`D2:=Today-25/24;`

`Test(D1,D2,1);`

`D2:=Today-26/24;`

`Test(D1,D2,5);`

`D2:=Today-5.4;`

`Test(D1,D2,5);`

`D2:=Today-2.5;`

`Test(D1,D2,1);`

`Test(D1,D2,2);`

`Test(D1,D2,3);`

End.

WithinPastHours

Synopsis: Check whether two datetimes are only a number of hours apart

Declaration: `function WithinPastHours(const ANow: TDateTime; const AThen: TDateTime;
 const AHours: Int64) : Boolean`

Visibility: `default`

Description: `WithinPastHours` compares the timestamps `ANow` and `AThen` and returns `True` if the difference between them is at most `AHours` hours apart, or `False` if they are further apart.

Remark: Since this function uses the `HoursBetween` (356) function to calculate the difference in Hours, this means that fractional hours do not count, and the fractional part is simply dropped, so for two dates actually 2 and a half hours apart, the result will also be `True`

See also: `WithinPastYears` (416), `WithinPastMonths` (413), `WithinPastWeeks` (415), `WithinPastDays` (409), `WithinPastMinutes` (412), `WithinPastSeconds` (414), `WithinPastMilliseconds` (411)

Listing: `./datutex/ex51.pp`

Program `Example51`;

{ This program demonstrates the WithinPastHours function }

Uses `SysUtils`, `DateUtils`;

Procedure `Test`(`ANow`, `AThen` : `TDateTime`; `AHours` : `Integer`);

begin

`Write`(`DateTimeToStr`(`AThen`), ' and ', `DateTimeToStr`(`ANow`));

`Write`(' are within ', `AHours`, ' hours: ');

`WriteLn`(`WithinPastHours`(`ANow`, `AThen`, `AHours`));

end;

Var

`D1`, `D2` : `TDateTime`;

Begin

`D1:=Now`;

`D2:=D1-(59*OneMinute)`;

`Test`(`D1`, `D2`, 1);

`D2:=D1-(61*OneMinute)`;

`Test`(`D1`, `D2`, 1);

`D2:=D1-(122*OneMinute)`;

`Test`(`D1`, `D2`, 1);

`D2:=D1-(306*OneMinute)`;

`Test`(`D1`, `D2`, 5);

`D2:=D1-(5.4*OneHour)`;

`Test`(`D1`, `D2`, 5);

`D2:=D1-(2.5*OneHour)`;

`Test`(`D1`, `D2`, 1);

`Test`(`D1`, `D2`, 2);

`Test`(`D1`, `D2`, 3);

End.

WithinPastMilliseconds

Synopsis: Check whether two datetimes are only a number of milliseconds apart

Declaration: `function WithinPastMilliseconds(const ANow: TDateTime;`
`const AThen: TDateTime;`
`const AMilliseconds: Int64) : Boolean`

Visibility: `default`

Description: `WithinPastMilliseconds` compares the timestamps `ANow` and `AThen` and returns `True` if the difference between them is at most `AMilliseconds` milliseconds apart, or `False` if they are further apart.

Remark: Since this function uses the `MillisecondsBetween` (372) function to calculate the difference in milliseconds, this means that fractional milliseconds do not count, and the fractional part is simply dropped, so for two dates actually 2 and a half milliseconds apart, the result will also be `True`

See also: `WithinPastYears` (416), `WithinPastMonths` (413), `WithinPastWeeks` (415), `WithinPastDays` (409), `WithinPastHours` (410), `WithinPastMinutes` (412), `WithinPastSeconds` (414)

Listing: `./datutex/ex54.pp`

Program `Example54`;

{ This program demonstrates the WithinPastMilliseconds function }

Uses `SysUtils, DateUtils`;

Procedure `Test(ANow, AThen : TDateTime; AMilliseconds : Integer);`

begin

`Write(TimeToStr(AThen), ' and ', TimeToStr(ANow));`

`Write(' are within ', AMilliseconds, ' milliseconds: ');`

`WriteLn(WithinPastMilliseconds(ANow, AThen, AMilliseconds));`

end;

Var

`D1, D2 : TDateTime;`

Begin

`D1:=Now;`

`D2:=D1-(0.9*OneMilliSecond);`

`Test(D1, D2, 1);`

`D2:=D1-(1.0*OneMilliSecond);`

`Test(D1, D2, 1);`

`D2:=D1-(1.1*OneMilliSecond);`

`Test(D1, D2, 1);`

`D2:=D1-(2.5*OneMilliSecond);`

`Test(D1, D2, 1);`

`Test(D1, D2, 2);`

`Test(D1, D2, 3);`

End.

WithinPastMinutes

Synopsis: Check whether two datetimes are only a number of minutes apart

Declaration: `function WithinPastMinutes(const ANow: TDateTime; const AThen: TDateTime;
const AMinutes: Int64) : Boolean`

Visibility: `default`

Description: `WithinPastMinutes` compares the timestamps `ANow` and `AThen` and returns `True` if the difference between them is at most `AMinutes` minutes apart, or `False` if they are further apart.

Remark: Since this function uses the `MinutesBetween` (375) function to calculate the difference in Minutes, this means that fractional minutes do not count, and the fractional part is simply dropped, so for two dates actually 2 and a half minutes apart, the result will also be `True`

See also: `WithinPastYears` (416), `WithinPastMonths` (413), `WithinPastWeeks` (415), `WithinPastDays` (409), `WithinPastHours` (410), `WithinPastSeconds` (414), `WithinPastMilliseconds` (411)

Listing: ./datutex/ex52.pp

Program Example52;

{ This program demonstrates the WithinPastMinutes function }

Uses SysUtils, DateUtils;

Procedure Test(ANow, AThen : TDateTime; AMinutes : Integer);

begin

Write(DateTimeToStr(AThen), ' and ', DateTimeToStr(ANow));

Write(' are within ', AMinutes, ' Minutes: ');

WriteLn(WithinPastMinutes(ANow, AThen, AMinutes));

end;

Var

 D1, D2 : TDateTime;

Begin

 D1:=Now;

 D2:=D1-(59*OneSecond);

 Test(D1, D2, 1);

 D2:=D1-(61*OneSecond);

 Test(D1, D2, 1);

 D2:=D1-(122*OneSecond);

 Test(D1, D2, 1);

 D2:=D1-(306*OneSecond);

 Test(D1, D2, 5);

 D2:=D1-(5.4*OneMinute);

 Test(D1, D2, 5);

 D2:=D1-(2.5*OneMinute);

 Test(D1, D2, 1);

 Test(D1, D2, 2);

 Test(D1, D2, 3);

End.

WithinPastMonths

Synopsis: Check whether two datetimes are only a number of months apart

Declaration: function WithinPastMonths(const ANow: TDateTime; const AThen: TDateTime;
 const AMonths: Integer) : Boolean

Visibility: default

Description: WithinPastMonths compares the timestamps ANow and AThen and returns True if the difference between them is at most AMonths months apart, or False if they are further apart.

Remark: Since this function uses the MonthsBetween (378) function to calculate the difference in Months, this means that fractional months do not count, and the fractional part is simply dropped, so for two dates actually 2 and a half months apart, the result will also be True

See also: WithinPastYears (416), WithinPastWeeks (415), WithinPastDays (409), WithinPastHours (410), WithinPastMinutes (412), WithinPastSeconds (414), WithinPastMilliseconds (411)

Listing: ./datutex/ex48.pp

Program Example48;

{ This program demonstrates the WithinPastMonths function }

Uses SysUtils, DateUtils;

Procedure Test(ANow, AThen : TDateTime; AMonths : Integer);

begin

Write(**DateToStr**(AThen), ' and ', **DateToStr**(ANow));

Write(' are within ', AMonths, ' months: ');

WriteLn(WithinPastMonths (ANow, AThen, AMonths));

end;

Var

 D1, D2 : TDateTime;

Begin

 D1:=Today;

 D2:=Today-364;

 Test(D1,D2,12);

 D2:=Today-365;

 Test(D1,D2,12);

 D2:=Today-366;

 Test(D1,D2,12);

 D2:=Today-390;

 Test(D1,D2,12);

 D2:=Today-368;

 Test(D1,D2,11);

 D2:=Today-1000;

 Test(D1,D2,31);

 Test(D1,D2,32);

 Test(D1,D2,33);

End.

WithinPastSeconds

Synopsis: Check whether two datetimes are only a number of seconds apart

Declaration: function WithinPastSeconds(const ANow: TDateTime; const AThen: TDateTime;
 const ASeconds: Int64) : Boolean

Visibility: default

Description: WithinPastSeconds compares the timestamps ANow and AThen and returns True if the difference between them is at most ASeconds seconds apart, or False if they are further apart.

Remark: Since this function uses the SecondsBetween (392) function to calculate the difference in seconds, this means that fractional seconds do not count, and the fractional part is simply dropped, so for two dates actually 2 and a half seconds apart, the result will also be True

See also: WithinPastYears (416), WithinPastMonths (413), WithinPastWeeks (415), WithinPastDays (409), WithinPastHours (410), WithinPastMinutes (412), WithinPastMilliseconds (411)

Listing: ./datutex/ex53.pp

Program Example53;

{ This program demonstrates the WithinPastSeconds function }

Uses SysUtils, DateUtils;

Procedure Test(ANow, AThen : TDateTime; ASeconds : Integer);

begin

Write(DateTimeToStr(AThen), ' and ', DateTimeToStr(ANow));

Write(' are within ', ASeconds, ' seconds: ');

WriteLn(WithinPastSeconds(ANow, AThen, ASeconds));

end;

Var

 D1, D2 : TDateTime;

Begin

 D1:=Now;

 D2:=D1-(999*OneMilliSecond);

 Test(D1, D2, 1);

 D2:=D1-(1001*OneMilliSecond);

 Test(D1, D2, 1);

 D2:=D1-(2001*OneMilliSecond);

 Test(D1, D2, 1);

 D2:=D1-(5001*OneMilliSecond);

 Test(D1, D2, 5);

 D2:=D1-(5.4*OneSecond);

 Test(D1, D2, 5);

 D2:=D1-(2.5*OneSecond);

 Test(D1, D2, 1);

 Test(D1, D2, 2);

 Test(D1, D2, 3);

End.

WithinPastWeeks

Synopsis: Check whether two datetimes are only a number of weeks apart

Declaration: function WithinPastWeeks(const ANow: TDateTime; const AThen: TDateTime;
 const AWeeks: Integer) : Boolean

Visibility: default

Description: WithinPastWeeks compares the timestamps ANow and AThen and returns True if the difference between them is at most AWeeks weeks apart, or False if they are further apart.

Remark: Since this function uses the WeeksBetween (407) function to calculate the difference in Weeks, this means that fractional Weeks do not count, and the fractional part is simply dropped, so for two dates actually 2 and a half weeks apart, the result will also be True

See also: WithinPastYears (416), WithinPastMonths (413), WithinPastDays (409), WithinPastHours (410), WithinPastMinutes (412), WithinPastSeconds (414), WithinPastMilliseconds (411)

Listing: ./datutex/ex49.pp

Program Example49;

{ This program demonstrates the WithinPastWeeks function }

Uses SysUtils, DateUtils;

Procedure Test(ANow, AThen : TDateTime; AWeeks : Integer);

begin

Write(**DateToStr**(AThen), ' and ', **DateToStr**(ANow));

Write(' are within ', AWeeks, ' weeks: ');

WriteLn(WithinPastWeeks(ANow, AThen, AWeeks));

end;

Var

 D1, D2 : TDateTime;

Begin

 D1:=Today;

 D2:=Today-7;

 Test(D1, D2, 1);

 D2:=Today-8;

 Test(D1, D2, 1);

 D2:=Today-14;

 Test(D1, D2, 1);

 D2:=Today-35;

 Test(D1, D2, 5);

 D2:=Today-36;

 Test(D1, D2, 5);

 D2:=Today-17;

 Test(D1, D2, 1);

 Test(D1, D2, 2);

 Test(D1, D2, 3);

End.

WithinPastYears

Synopsis: Check whether two datetimes are only a number of years apart

Declaration: function WithinPastYears(const ANow: TDateTime; const AThen: TDateTime;
 const AYears: Integer) : Boolean

Visibility: default

Description: WithinPastYears compares the timestamps ANow and AThen and returns True if the difference between them is at most AYears years apart, or False if they are further apart.

Remark: Since this function uses the YearsBetween (418) function to calculate the difference in years, this means that fractional years do not count, and the fractional part is simply dropped, so for two dates actually 2 and a half years apart, the result will also be True

See also: WithinPastMonths (413), WithinPastWeeks (415), WithinPastDays (409), WithinPastHours (410), WithinPastMinutes (412), WithinPastSeconds (414), WithinPastMilliseconds (411)

Listing: ./datutex/ex47.pp

Program Example47;

{ This program demonstrates the WithinPastYears function }

Uses SysUtils, DateUtils;

Procedure Test(ANow, AThen : TDateTime; AYears : Integer);

begin

Write(**DateToStr**(AThen), ' and ', **DateToStr**(ANow));

Write(' are within ', AYears, ' years: ');

WriteLn(WithinPastYears(ANow, AThen, AYears));

end;

Var

 D1, D2 : TDateTime;

Begin

 D1:=Today;

 D2:=Today-364;

 Test(D1, D2, 1);

 D2:=Today-365;

 Test(D1, D2, 1);

 D2:=Today-366;

 Test(D1, D2, 1);

 D2:=Today-390;

 Test(D1, D2, 1);

 D2:=Today-368;

 Test(D1, D2, 1);

 D2:=Today-1000;

 Test(D1, D2, 1);

 Test(D1, D2, 2);

 Test(D1, D2, 3);

End.

YearOf

Synopsis: Extract the year from a given date.

Declaration: function YearOf(const AValue: TDateTime) : Word

Visibility: default

Description: YearOf returns the year part of the AValue date/time indication. It is a number between 1 and 9999.

See also: MonthOf ([377](#)), DayOf ([339](#)), WeekOf ([405](#)), HourOf ([354](#)), MinuteOf ([373](#)), SecondOf ([390](#)), MilliSecondOf ([369](#))

Listing: ./datutex/ex23.pp

Program Example23;

{ This program demonstrates the YearOf function }

Uses SysUtils, DateUtils;

```

Var
  D : TDateTime;

Begin
  D:=Now;
  WriteLn ( 'Year      : ', YearOf(D));
  WriteLn ( 'Month     : ', MonthOf(D));
  WriteLn ( 'Day       : ', DayOf(D));
  WriteLn ( 'Week      : ', WeekOf(D));
  WriteLn ( 'Hour      : ', HourOf(D));
  WriteLn ( 'Minute    : ', MinuteOf(D));
  WriteLn ( 'Second    : ', SecondOf(D));
  WriteLn ( 'MilliSecond : ', MilliSecondOf(D));
End.

```

YearsBetween

Synopsis: Calculate the number of whole years between two DateTime values

Declaration: `function YearsBetween(const ANow: TDateTime;const AThen: TDateTime) : Integer`

Visibility: default

Description: `YearsBetween` returns the number of whole years between `ANow` and `AThen`. This number is an approximation, based on an average number of days of 365.25 per year (average over 4 years). This means the fractional part of a year is dropped.

See also: `MonthsBetween` ([378](#)), `WeeksBetween` ([407](#)), `DaysBetween` ([341](#)), `HoursBetween` ([356](#)), `MinutesBetween` ([375](#)), `SecondsBetween` ([392](#)), `MillisecondsBetween` ([372](#)), `YearSpan` ([419](#))

Listing: `./datutex/ex55.pp`

Program Example55;

{ This program demonstrates the YearsBetween function }

Uses SysUtils, DateUtils;

Procedure Test(ANow, AThen : TDateTime);

```

begin
  Write('Number of years between ');
  Write(DateToStr(AThen), ' and ', DateToStr(ANow));
  WriteLn(' : ', YearsBetween(ANow, AThen));
end;

```

```

Var
  D1,D2 : TDateTime;

```

```

Begin
  D1:=Today;
  D2:=Today-364;
  Test(D1,D2);
  D2:=Today-365;
  Test(D1,D2);
  D2:=Today-366;

```

```

Test(D1,D2);
D2:=Today-390;
Test(D1,D2);
D2:=Today-368;
Test(D1,D2);
D2:=Today-1000;
Test(D1,D2);
End.

```

YearSpan

Synopsis: Calculate the approximate number of years between two DateTime values.

Declaration: `function YearSpan(const ANow: TDateTime;const AThen: TDateTime) : Double`

Visibility: default

Description: YearSpan returns the number of years between ANow and AThen, including any fractional parts of a year. This number is an approximation, based on an average number of days of 365.25 per year (average over 4 years).

See also: MonthSpan ([379](#)), WeekSpan ([409](#)), DaySpan ([344](#)), HourSpan ([357](#)), MinuteSpan ([376](#)), SecondSpan ([393](#)), MilliSecondSpan ([372](#)), YearsBetween ([418](#))

Listing: ./datutex/ex63.pp

Program Example63;

{ This program demonstrates the YearSpan function }

Uses SysUtils, DateUtils;

Procedure Test(ANow, AThen : TDateTime);

begin

Write('Number of years between ');

Write(DateToStr(AThen), ' and ', DateToStr(ANow));

WriteLn(' : ', YearSpan(ANow, AThen));

end;

Var

D1, D2 : TDateTime;

Begin

D1:=Today;

D2:=Today-364;

Test(D1,D2);

D2:=Today-365;

Test(D1,D2);

D2:=Today-366;

Test(D1,D2);

D2:=Today-390;

Test(D1,D2);

D2:=Today-368;

Test(D1,D2);

D2:=Today-1000;

Test(D1,D2);

End.

Yesterday

Synopsis: Return the previous day.

Declaration: `function Yesterday : TDateTime`

Visibility: default

Description: `Yesterday` returns yesterday's date. `Yesterday` is determined from the system clock, i.e. it is `Today` (399) -1.

See also: `Today` (399), `Tomorrow` (399)

Listing: `./datutex/ex18.pp`

Program `Example18;`

{ This program demonstrates the Yesterday function }

Uses `SysUtils , DateUtils ;`

Begin

`WriteLn (FormatDateTime('"Today is " dd mmm yyyy ',Today));`

`WriteLn (FormatDateTime('"Yesterday was " dd mmm yyyy ',Yesterday));`

End.

Chapter 5

Reference for unit 'Dos'

5.1 System information

Functions for retrieving and setting general system information such as date and time.

Table 5.1:

Name	Description
DosVersion (430)	Get OS version
GetCBreak (435)	Get setting of control-break handling flag
GetDate (435)	Get system date
GetIntVec (438)	Get interrupt vector status
GetTime (439)	Get system time
GetVerify (440)	Get verify flag
Intr (440)	Execute an interrupt
Keep (441)	Keep process in memory and exit
MSDos (441)	Execute MS-dos function call
PackTime (441)	Pack time for file time
SetCBreak (442)	Set control-break handling flag
SetDate (442)	Set system date
SetIntVec (443)	Set interrupt vectors
SetTime (444)	Set system time
SetVerify (444)	Set verify flag
SwapVectors (444)	Swap interrupt vectors
UnPackTime (445)	Unpack file time

5.2 Process handling

Functions to handle process information and starting new processes.

5.3 Directory and disk handling

Routines to handle disk information.

Table 5.2:

Name	Description
DosExitCode (429)	Exit code of last executed program
EnvCount (431)	Return number of environment variables
EnvStr (431)	Return environment string pair
Exec (432)	Execute program
GetEnv (436)	Return specified environment string

Table 5.3:

Name	Description
AddDisk (427)	Add disk to list of disks (UNIX only)
DiskFree (428)	Return size of free disk space
DiskSize (429)	Return total disk size

5.4 File handling

Routines to handle files on disk.

Table 5.4:

Name	Description
FExpand (432)	Expand filename to full path
FindClose (432)	Close finfirst/findnext session
FindFirst (433)	Start find of file
FindNext (434)	Find next file
FSearch (434)	Search for file in a path
FSplit (434)	Split filename in parts
GetFAttr (436)	Return file attributes
GetFTime (437)	Return file time
GetLongName (438)	Convert short filename to long filename (DOS only)
GetShortName (439)	Convert long filename to short filename (DOS only)
SetFAttr (443)	Set file attributes
SetFTime (443)	Set file time

5.5 File open mode constants.

These constants are used in the Mode field of the TextRec record. Gives information on the file-mode of the text I/O. For their definitions consult the following table:

5.6 File attributes

The File Attribute constants are used in FindFirst (433), FindNext (434) to determine what type of special file to search for in addition to normal files. These flags are also used in the SetFAttr (443) and

Table 5.5: Possible mode constants

Constant	Description	Value
fmclosed	File is closed	\$D7B0
fminput	File is read only	\$D7B1
fmoutput	File is write only	\$D7B2
fminout	File is read and write	\$D7B3

GetFAttr (436) routines to set and retrieve attributes of files. For their definitions consult fileattributes (422).

Table 5.6: Possible file attributes

Constant	Description	Value
readonly	Read-Only file attribute	\$01
hidden	Hidden file attribute	\$02
sysfile	System file attribute	\$04
volumeid	Volumd ID file attribute	\$08
directory	Directory file attribute	\$10
archive	Archive file attribute	\$20
anyfile	Match any file attribute	\$3F

5.7 Overview

The DOS unit gives access to some operating system calls related to files, the file system, date and time. Except for the PalmOS target, this unit is available to all supported platforms.

The unit was first written for dos by Florian Klaempfl. It was ported to linux by Mark May and enhanced by Michael Van Canneyt. The Amiga version was ported by Nils Sjöholm.

Under non-DOS systems, some of the functionality is lost, as it is either impossible or meaningless to implement it. Other than that, the functionality for all operating systems is the same.

5.8 Constants, types and variables

Constants

`anyfile = $3F`

Match any file attribute

`archive = $20`

Archive file attribute

`directory = $10`

Directory file attribute

`fauxiliary = $0010`

CPU auxiliary flag. Not used.

fcarry = \$0001

CPU carry flag. Not used.

FileNameLen = 255

Maximum length of a filename

filerecnamelength = 255

Maximum length of FileName part in FileRec ([425](#))

fmclosed = \$D7B0

File is closed

fminout = \$D7B3

File is read and write

fminput = \$D7B1

File is read only

fmoutput = \$D7B2

File is write only

foverflow = \$0800

CPU overflow flag. Not used.

fparity = \$0004

CPU parity flag. Not used.

fsign = \$0080

CPU sign flag. Not used.

fzero = \$0040

CPU zero flag. Not used.

hidden = \$02

Hidden file attribute

readonly = \$01

Read-Only file attribute

sysfile = \$04

System file attribute

TextRecBufSize = 256

Size of default buffer in TextRec ([427](#))

TextRecNameLength = 256

Maximum length of filename in TextRec ([427](#))

volumeid = \$08

Volumd ID file attribute

Types

ComStr =

Command-line string type

DateTime = packed record

```
Year : Word;
Month : Word;
Day : Word;
Hour : Word;
Min : Word;
Sec : Word;
```

end

The DateTime type is used in PackTime ([441](#)) and UnPackTime ([445](#)) for setting/reading file times with GetFTime ([437](#)) and SetFTime ([443](#)).

DirStr =

Full directory string type.

ExtStr =

Filename extension string type.

FileRec = packed record

```
Handle : THandle;
Mode : LongInt;
RecSize : SizeInt;
_private : Array[1..3*SizeOf(SizeInt)+5*SizeOf(pointer)] of Byte;
UserData : Array[1..16] of Byte;
name : Array[0..filerecnamelength] of Char;
```

end

FileRec is used for internal representation of typed and untyped files.

NameStr =

Fill filename string type.

```
PathStr =
```

Full File path string type.

```
Registers = packed record
end
```

Record to keep CPU registers for MSDos (441) call. Unused.

```
SearchRec = packed record
  SearchNum : LongInt;
  SearchPos : LongInt;
  DirPtr : Pointer;
  SearchType : Byte;
  SearchAttr : Byte;
  Fill : Array[1..07] of Byte;
  Attr : Byte;
  Time : LongInt;
  Size : LongInt;
  Reserved : Word;
  Name : String;
  SearchSpec : String;
  NamePos : Word;
end
```

SearchRec is filled by the FindFirst (433) call and can be used in subsequent FindNext (434) calls to search for files. The structure of this record depends on the platform. Only the following fields are present on all platforms:

Attr File attributes.

Time File modification time.

Size File size

Name File name (name part only, no path)

```
TextBuf = Array[0..TextRecBufSize-1] of Char
```

Type for default buffer in TextRec (427)

```
TextRec = packed record
  Handle : THandle;
  Mode : LongInt;
  bufsize : SizeInt;
  LineEnd : TLineEndStr;
  bufpos : SizeInt;
  bufend : SizeInt;
  bufptr : ^TextBuf;
  openfunc : pointer;
  inoutfunc : pointer;
```

```

flushfunc : pointer;
closefunc : pointer;
UserData : Array[1..16] of Byte;
name : Array[0..textrecnamelength-1] of Char;
buffer : TextBuf;
end

```

TextRec describes the internal working of a Text file.

Remark that this is not binary compatible with the Turbo Pascal definition of TextRec, since the sizes of the different fields are different.

TLineEndStr =

Variables

DosError : Integer

The DosError variable is used by the procedures in the dos unit to report errors. It can have the following values :

Table 5.7: Dos error codes

Value	Meaning
2	File not found.
3	path not found.
5	Access denied.
6	Invalid handle.
8	Not enough memory.
10	Invalid environment.
11	Invalid format.
18	No more files.

Other values are possible, but are not documented.

5.9 Procedures and functions

AddDisk

Synopsis: Add disk definition to list if drives (Unix only)

Declaration: procedure AddDisk(const path: String)

Visibility: default

Description: AddDisk adds a filename *S* to the internal list of disks. It is implemented for systems which do not use DOS type drive letters. This list is used to determine which disks to use in the DiskFree (428) and DiskSize (429) calls. The DiskFree (428) and DiskSize (429) functions need a file on the specified drive, since this is required for the `statfs` system call. The names are added sequentially. The dos initialization code presets the first three disks to:

- ' . ' for the current drive,
- ' /fd0 / . ' for the first floppy-drive (linux only).
- ' /fd1 / . ' for the second floppy-drive (linux only).
- ' / ' for the first hard disk.

The first call to `AddDisk` will therefore add a name for the second harddisk, The second call for the third drive, and so on until 23 drives have been added (corresponding to drives 'D:' to 'Z:')

Errors: None

See also: `DiskFree` ([428](#)), `DiskSize` ([429](#))

DiskFree

Synopsis: Get free size on Disk.

Declaration: `function DiskFree(drive: Byte) : Int64`

Visibility: default

Description: `DiskFree` returns the number of free bytes on a disk. The parameter `Drive` indicates which disk should be checked. This parameter is 1 for floppy a:, 2 for floppy b:, etc. A value of 0 returns the free space on the current drive.

Remark: For Unices: The `diskfree` and `disksize` functions need a file on the specified drive, since this is required for the `statfs` system call. These filenames are set in the initialization of the dos unit, and have been preset to :

- ' . ' for the current drive,
- ' /fd0 / . ' for the first floppy-drive (linux only).
- ' /fd1 / . ' for the second floppy-drive (linux only).
- ' / ' for the first hard disk.

There is room for 1-26 drives. You can add a drive with the `AddDisk` ([427](#)) procedure. These settings can be coded in `dos.pp`, in the initialization part.

Errors: -1 when a failure occurs, or an invalid drive number is given.

See also: `DiskSize` ([429](#)), `AddDisk` ([427](#))

Listing: `./dosex/ex6.pp`

Program Example6;
uses Dos;

{ Program to demonstrate the DiskSize and DiskFree function. }

begin
 WriteLn('This partition size has ', **DiskSize**(0), ' bytes');
 WriteLn('Currently ', **DiskFree**(0), ' bytes are free');
end.

DiskSize

Synopsis: Get total size of disk.

Declaration: `function DiskSize(drive: Byte) : Int64`

Visibility: default

Description: `DiskSize` returns the total size (in bytes) of a disk. The parameter `Drive` indicates which disk should be checked. This parameter is 1 for floppy a:, 2 for floppy b:, etc. A value of 0 returns the size of the current drive.

Remark: For unix only: The `diskfree` and `disksize` functions need a file on the specified drive, since this is required for the `statfs` system call. These filenames are set in the initialization of the dos unit, and have been preset to :

- ' .' for the current drive,
- '/fd0/ .' for the first floppy-drive (linux only).
- '/fd1/ .' for the second floppy-drive (linux only).
- '/' for the first hard disk.

There is room for 1-26 drives. You can add a drive with the `AddDisk` (427) procedure. These settings can be coded in `dos.pp`, in the initialization part.

For an example, see `DiskFree` (428).

Errors: -1 when a failure occurs, or an invalid drive number is given.

See also: `DiskFree` (428), `AddDisk` (427)

DosExitCode

Synopsis: Exit code of last executed program.

Declaration: `function DosExitCode : Word`

Visibility: default

Description: `DosExitCode` contains (in the low byte) the exit-code of a program executed with the `Exec` call.

Errors: None.

See also: `Exec` (432)

Listing: `./dosex/ex5.pp`

```

Program Example5;
uses Dos;

{ Program to demonstrate the Exec and DosExitCode function. }

begin
  {$IFDEF Unix}
    WriteLn( 'Executing /bin/ls -la ');
    Exec( '/bin/ls ', '-la ');
  {$ELSE}
    WriteLn( 'Executing Dir ');
    Exec( GetEnv( 'COMSPEC' ), '/C dir ');
  {$ENDIF}
  WriteLn( 'Program returned with ExitCode ', Lo(DosExitCode));
end.

```

DosVersion

Synopsis: Current OS version

Declaration: `function DosVersion : Word`

Visibility: default

Description: `DosVersion` returns the operating system or kernel version. The low byte contains the major version number, while the high byte contains the minor version number.

Remark: On systems where versions consists of more then two numbers, only the first two numbers will be returned. For example Linux version 2.1.76 will give you `DosVersion` 2.1. Some operating systems, such as FreeBSD, do not have system calls to return the kernel version, in that case a value of 0 will be returned.

Errors: None.

Listing: `./dosex/ex1.pp`

```

Program Example1;
uses Dos;

{ Program to demonstrate the DosVersion function. }

var
    OS      : string[32];
    Version : word;
begin
    {$IFDEF LINUX}
        OS:= 'Linux';
    {$ENDIF}
    {$ifdef FreeBSD}
        OS:= 'FreeBSD';
    {$endif}
    {$ifdef NetBSD}
        OS:= 'NetBSD';
    {$endif}
    {$ifdef Solaris}
        OS:= 'Solaris';
    {$endif}
    {$ifdef QNX}
        OS:= 'QNX';
    {$endif}

    {$IFDEF DOS}
        OS:= 'Dos';
    {$ENDIF}
    Version:=DosVersion;
    WriteLn('Current ',OS,' version is ',Lo(Version),'.',Hi(Version));
end.

```

DTToUnixDate

Synopsis: Convert a DateTime to unix timestamp

Declaration: `function DTToUnixDate(DT: DateTime) : LongInt`

Visibility: default

Description: `DTToUnixDate` converts the `DateTime` value in `DT` to a unix timestamp. It is an internal function, implemented on Unix platforms, and should not be used.

Errors: None.

See also: `UnixDateToDT` ([445](#)), `PackTime` ([441](#)), `UnpackTime` ([445](#)), `GetTime` ([439](#)), `SetTime` ([444](#))

EnvCount

Synopsis: Return the number of environment variables

Declaration: `function EnvCount : LongInt`

Visibility: default

Description: `EnvCount` returns the number of environment variables.

Errors: None.

See also: `EnvStr` ([431](#)), `GetEnv` ([436](#))

EnvStr

Synopsis: Return environment variable by index

Declaration: `function EnvStr(Index: LongInt) : String`

Visibility: default

Description: `EnvStr` returns the `Index`-th `Name=Value` pair from the list of environment variables. The index of the first pair is zero.

Errors: The length is limited to 255 characters.

See also: `EnvCount` ([431](#)), `GetEnv` ([436](#))

Listing: `./dosex/ex13.pp`

Program Example13;

uses Dos;

{ Program to demonstrate the EnvCount and EnvStr function. }

var

i : Longint;

begin

WriteLn('Current Environment is: ');

for *i* := 1 **to** EnvCount **do**

WriteLn(EnvStr(*i*));

end.

Exec

Synopsis: Execute another program, and wait for it to finish.

Declaration: `procedure Exec(const path: PathStr;const comline: ComStr)`

Visibility: default

Description: Exec executes the program in Path, with the options given by ComLine.

After the program has terminated, the procedure returns. The Exit value of the program can be consulted with the `DosExitCode` function.

For an example, see `DosExitCode` (429)

Errors: Errors are reported in `DosError`.

See also: `DosExitCode` (429)

FExpand

Synopsis: Expand a relative path to an absolute path

Declaration: `function FExpand(const path: PathStr) : PathStr`

Visibility: default

Description: FExpand takes its argument and expands it to a complete filename, i.e. a filename starting from the root directory of the current drive, prepended with the drive-letter or volume name (when supported).

Remark: On case sensitive file systems (such as unix and linux), the resulting name is left as it is, otherwise it is converted to uppercase.

Errors: `FSplit` (434)

Listing: `./dosex/ex5.pp`

```

Program Example5;
uses Dos;

{ Program to demonstrate the Exec and DosExitCode function. }

begin
  {$IFDEF Unix}
    WriteLn('Executing /bin/ls -la');
    Exec('/bin/ls', '-la');
  {$ELSE}
    WriteLn('Executing Dir');
    Exec(GetEnv('COMSPEC'), '/C dir');
  {$ENDIF}
  WriteLn('Program returned with ExitCode ', Lo(DosExitCode));
end.

```

FindClose

Synopsis: Dispose resources allocated by a `FindFirst` (433)/`FindNext` (434) sequence.

Declaration: `procedure FindClose(var f: SearchRec)`

Visibility: default

Description: FindClose frees any resources associated with the search record F.

This call is needed to free any internal resources allocated by the FindFirst (433) or FindNext (434) calls.

The unix implementation of the dos unit therefore keeps a table of open directories, and when the table is full, closes one of the directories, and reopens another. This system is adequate but slow if you use a lot of searchrecs.

So, to speed up the findfirst/findnext system, the FindClose call was implemented. When you don't need a searchrec any more, you can tell this to the dos unit by issuing a FindClose call. The directory which is kept open for this searchrec is then closed, and the table slot freed.

Remark: It is recommended to use the linux call Glob when looking for files on linux.

Errors: Errors are reported in DosError.

See also: FindFirst (433), FindNext (434)

FindFirst

Synopsis: Start search for one or more files.

Declaration: procedure FindFirst(const path: PathStr;attr: Word;var f: SearchRec)

Visibility: default

Description: FindFirst searches the file specified in Path. Normal files, as well as all special files which have the attributes specified in Attr will be returned.

It returns a SearchRec record for further searching in F. Path can contain the wildcard characters ? (matches any single character) and * (matches 0 ore more arbitrary characters). In this case FindFirst will return the first file which matches the specified criteria. If DosError is different from zero, no file(s) matching the criteria was(were) found.

Remark: On os/2, you cannot issue two different FindFirst calls. That is, you must close any previous search operation with FindClose (432) before starting a new one. Failure to do so will end in a Run-Time Error 6 (Invalid file handle)

Errors: Errors are reported in DosError.

See also: FindNext (434), FindClose (432)

Listing: ./dosex/ex7.pp

```

Program Example7;
uses Dos;

{ Program to demonstrate the FindFirst and FindNext function. }

var
  Dir : SearchRec;
begin
  FindFirst( '*.*', archive, Dir);
  WriteLn( 'FileName '+Space(32), 'FileSize ':9);
  while (DosError=0) do
    begin
      WriteLn( Dir.Name+Space(40-Length( Dir.Name)), Dir.Size:9);
      FindNext( Dir);
    end;
  FindClose( Dir );
end.

```

FindNext

Synopsis: Find next matching file after FindFirst ([433](#))

Declaration: `procedure FindNext(var f: SearchRec)`

Visibility: default

Description: FindNext takes as an argument a SearchRec from a previous FindNext call, or a FindFirst call, and tries to find another file which matches the criteria, specified in the FindFirst call. If DosError is different from zero, no more files matching the criteria were found.

For an example, see FindFirst ([433](#)).

Errors: DosError is used to report errors.

See also: FindFirst ([433](#)), FindClose ([432](#))

FSearch

Synopsis: Search a file in searchpath

Declaration: `function FSearch(path: PathStr;dirlist: String) : PathStr`

Visibility: default

Description: FSearch searches the file Path in all directories listed in DirList. The full name of the found file is returned. DirList must be a list of directories, separated by semi-colons. When no file is found, an empty string is returned.

Remark: On unix systems, DirList can also be separated by colons, as is customary on those environments.

Errors: None.

See also: FExpand ([432](#))

Listing: ./dosex/ex10.pp

```

Program Example10;
uses Dos;

{ Program to demonstrate the FSearch function. }

var
  s : string;
begin
  s:=FSearch(ParamStr(1),GetEnv('PATH'));
  if s='' then
    WriteLn(ParamStr(1),' not Found in PATH')
  else
    WriteLn(ParamStr(1),' Found in PATH at ',s);
end.

```

FSplit

Synopsis: Split a full-path filename in parts.

Declaration: `procedure FSplit(path: PathStr;var dir: DirStr;var name: NameStr;var ext: ExtStr)`

Visibility: default

Description: `FSplit` splits a full file name into 3 parts : A Path, a Name and an extension (in ext.) The extension is taken to be all letters after the *last* dot (.). For dos, however, an exception is made when `LFNSupport=False`, then the extension is defined as all characters after the *first* dot.

Errors: None.

See also: `FSearch` ([434](#))

Listing: ./dosex/ex12.pp

```

Program Example12;
uses Dos;

{ Program to demonstrate the FSplit function. }

var
    Path,Name,Ext : string;
begin
    FSplit(ParamStr(1),Path,Name,Ext);
    WriteLn( 'Splitted ',ParamStr(1), ' in: ');
    WriteLn( 'Path      : ',Path);
    WriteLn( 'Name      : ',Name);
    WriteLn( 'Extension : ',Ext);
end.

```

GetCBreak

Synopsis: Get control-Break flag

Declaration: `procedure GetCBreak(var breakvalue: Boolean)`

Visibility: default

Description: `GetCBreak` gets the status of CTRL-Break checking under dos and Amiga. When `BreakValue` is false, then dos only checks for the CTRL-Break key-press when I/O is performed. When it is set to True, then a check is done at every system call.

Remark: Under non-dos and non-Amiga operating systems, `BreakValue` always returns True.

Errors: None

See also: `SetCBreak` ([442](#))

GetDate

Synopsis: Get the current date

Declaration: `procedure GetDate(var year: Word;var month: Word;var mday: Word;
var wday: Word)`

Visibility: default

Description: `GetDate` returns the system's date. Year is a number in the range 1980..2099.mday is the day of the month, wday is the day of the week, starting with Sunday as day 0.

Errors: None.

See also: [GetTime \(439\)](#), [SetDate \(442\)](#)

Listing: ./dosex/ex2.pp

```

Program Example2;
uses Dos;

{ Program to demonstrate the GetDate function. }

const
  DayStr: array [0..6] of string [3] = ( 'Sun', 'Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat' );
  MonthStr: array [1..12] of string [3] = ( 'Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun',
                                           'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec' );

var
  Year, Month, Day, WDay : word;
begin
  GetDate(Year, Month, Day, WDay);
  WriteLn( 'Current date' );
  WriteLn( DayStr[WDay], ' ', ' ', Day, ' ', MonthStr[Month], ' ', Year, ' ' );
end.

```

GetEnv

Synopsis: Get environment variable by name.

Declaration: `function GetEnv(envvar: String) : String`

Visibility: default

Description: `Getenv` returns the value of the environment variable `EnvVar`. When there is no environment variable `EnvVar` defined, an empty string is returned.

Remark: Under some operating systems (such as unix), case is important when looking for `EnvVar`.

Errors: None.

See also: [EnvCount \(431\)](#), [EnvStr \(431\)](#)

Listing: ./dosex/ex14.pp

```

Program Example14;
uses Dos;

{ Program to demonstrate the GetEnv function. }

begin
  WriteLn( 'Current PATH is ', GetEnv( 'PATH' ) );
end.

```

GetFAttr

Synopsis: Get file attributes

Declaration: `procedure GetFAttr(var f; var attr: Word)`

Visibility: default

Description: `GetFAttr` returns the file attributes of the file-variable `f`. `F` can be a untyped or typed file, or of type `Text`. `f` must have been assigned, but not opened. The attributes can be examined with the following constants :

- `ReadOnly`
- `Hidden`
- `SysFile`
- `VolumeId`
- `Directory`
- `Archive`

Under linux, supported attributes are:

- `Directory`
- `ReadOnly` if the current process doesn't have access to the file.
- `Hidden` for files whose name starts with a dot (`'.'`).

Errors: Errors are reported in `DosError`

See also: `SetFAttr` ([443](#))

Listing: `./dosex/ex8.pp`

```

Program Example8;
uses Dos;

{ Program to demonstrate the GetFAttr function. }

var
  Attr : Word;
  f     : File;
begin
  Assign(f, ParamStr(1));
  GetFAttr(f, Attr);
  WriteLn('File ', ParamStr(1), ' has attribute ', Attr);
  if (Attr and archive) <> 0 then WriteLn(' - Archive ');
  if (Attr and directory) <> 0 then WriteLn(' - Directory ');
  if (Attr and readonly) <> 0 then WriteLn(' - Read-Only ');
  if (Attr and sysfile) <> 0 then WriteLn(' - System ');
  if (Attr and hidden) <> 0 then WriteLn(' - Hidden ');
end.

```

GetFTime

Synopsis: Get file last modification time.

Declaration: `procedure GetFTime(var f; var time: LongInt)`

Visibility: default

Description: `GetFTime` returns the modification time of a file. This time is encoded and must be decoded with `UnPackTime`. `F` must be a file type, which has been assigned, and opened.

Errors: Errors are reported in `DosError`

See also: SetFTime ([443](#)), PackTime ([441](#)), UnPackTime ([445](#))

Listing: ./dosex/ex9.pp

```

Program Example9;
uses Dos;

{ Program to demonstrate the GetFTime function. }

Function L0(w: word): string;
var
  s : string;
begin
  Str(w,s);
  if w<10 then
    L0:= '0'+s
  else
    L0:=s;
end;

var
  f      : File;
  Time   : Longint;
  DT     : DateTime;
begin
  if Paramcount>0 then
    Assign(f,ParamStr(1))
  else
    Assign(f,'ex9.pp');
  Reset(f);
  GetFTime(f,Time);
  Close(f);
  UnPackTime(Time,DT);
  Write('File ',ParamStr(1),' is last modified on ');
  Writeln(L0(DT.Month),'-',L0(DT.Day),'-',DT.Year,
          ' at ',L0(DT.Hour),':',L0(DT.Min));
end.

```

GetIntVec

Synopsis: Get interrupt vector

Declaration: `procedure GetIntVec(intno: Byte;var vector: pointer)`

Visibility: default

Description: GetIntVec returns the address of interrupt vector IntNo.

Remark: This call does nothing, it is present for compatibility only. Modern systems do not allow low level access to the hardware.

Errors: None.

See also: SetIntVec ([443](#))

GetLongName

Synopsis: Get the long filename of a DOS 8.3 filename.

Declaration: `function GetLongName(var p: String) : Boolean`

Visibility: default

Description: This function is only implemented in the GO32V2 version of Free Pascal.

`GetLongName` changes the filename `p` to a long filename if the dos call to do this is successful. The resulting string is the long file name corresponding to the short filename `p`.

The function returns `True` if the dos call was successful, `False` otherwise.

This function should only be necessary when using the DOS extender under Windows 95 and higher.

Errors: If the dos call was not successful, `False` is returned.

See also: `GetShortName` ([439](#))

GetMsCount

Declaration: `function GetMsCount : Int64`

Visibility: default

GetShortName

Synopsis: Get the short (8.3) filename of a long filename.

Declaration: `function GetShortName(var p: String) : Boolean`

Visibility: default

Description: This function is only implemented in the GO32V2 version of Free Pascal.

`GetShortName` changes the filename `p` to a short filename if the dos call to do this is successful. The resulting string is the short file name corresponding to the long filename `p`.

The function returns `True` if the dos call was successful, `False` otherwise.

This function should only be necessary when using the DOS extender under Windows 95 and higher.

Errors: If the dos call was not successful, `False` is returned.

See also: `GetLongName` ([438](#))

GetTime

Synopsis: Return the current time

Declaration: `procedure GetTime(var hour: Word; var minute: Word; var second: Word;
var sec100: Word)`

Visibility: default

Description: `GetTime` returns the system's time. Hour is on a 24-hour time scale. `sec100` is in hundredth of a second.

Remark: Certain operating systems (such as Amiga), always set the `sec100` field to zero.

Errors: None.

See also: `GetDate` ([435](#)), `SetTime` ([444](#))

Listing: ./dosex/ex3.pp

```

Program Example3;
uses Dos;

{ Program to demonstrate the GetTime function . }

Function L0(w: word): string;
var
  s : string;
begin
  Str(w,s);
  if w<10 then
    L0:= '0'+s
  else
    L0:=s;
end;

var
  Hour,Min,Sec,HSec : word;
begin
  GetTime(Hour,Min,Sec,HSec);
  WriteLn('Current time ');
  WriteLn(L0(Hour),':',L0(Min),':',L0(Sec));
end.

```

GetVerify

Synopsis: Get verify flag

Declaration: `procedure GetVerify(var verify: Boolean)`

Visibility: default

Description: `GetVerify` returns the status of the verify flag under dos. When `Verify` is `True`, then dos checks data which are written to disk, by reading them after writing. If `Verify` is `False`, then data written to disk are not verified.

Remark: Under non-dos systems (excluding os/2 applications running under vanilla DOS), `Verify` is always `True`.

Errors: None.

See also: `SetVerify` ([444](#))

Intr

Synopsis: Execute interrupt

Declaration: `procedure Intr(intno: Byte;var regs: Registers)`

Visibility: default

Description: `Intr` executes a software interrupt number `IntNo` (must be between 0 and 255), with processor registers set to `Regs`. After the interrupt call returned, the processor registers are saved in `Regs`.

Remark: Under non-dos operating systems, this call does nothing.

Errors: None.

See also: `MSDos` ([441](#))

Keep

Synopsis: Terminate and stay resident.

Declaration: `procedure Keep(exitcode: Word)`

Visibility: default

Description: `Keep` terminates the program, but stays in memory. This is used for TSR (Terminate Stay Resident) programs which catch some interrupt. `ExitCode` is the same parameter as the `Halt` function takes.

Remark: This call does nothing, it is present for compatibility only.

Errors: None.

MSDos

Synopsis: Execute MS-DOS system call

Declaration: `procedure MSDos(var regs: Registers)`

Visibility: default

Description: `MSDos` executes an operating system call. This is the same as doing a `Intr` call with the interrupt number for an os call.

Remark: Under non-dos operating systems, this call does nothing. On DOS systems, this calls interrupt \$21.

Errors: None.

See also: `Intr` ([440](#))

PackTime

Synopsis: Pack `DateTime` value to a packed-time format.

Declaration: `procedure PackTime(var t: DateTime; var p: LongInt)`

Visibility: default

Description: `UnPackTime` converts the date and time specified in `T` to a packed-time format which can be fed to `SetFTime`.

Errors: None.

See also: `SetFTime` ([443](#)), `FindFirst` ([433](#)), `FindNext` ([434](#)), `UnPackTime` ([445](#))

Listing: `./dosex/ex4.pp`

```

Program Example4;
uses Dos;

{ Program to demonstrate the PackTime and UnPackTime functions. }

var
    DT    : DateTime;
    Time  : longint;
begin
    with DT do
        begin
            Year:=1998;

```

```

    Month:=11;
    Day:=11;
    Hour:=11;
    Min:=11;
    Sec:=11;
end;
PackTime(DT,Time);
WriteLn('Packed Time : ',Time);
UnPackTime(Time,DT);
WriteLn('Unpacked Again: ');
with DT do
begin
    WriteLn('Year  ',Year);
    WriteLn('Month ',Month);
    WriteLn('Day   ',Day);
    WriteLn('Hour  ',Hour);
    WriteLn('Min   ',Min);
    WriteLn('Sec   ',Sec);
end;
end.

```

SetCBreak

Synopsis: Set Control-Break flag status

Declaration: `procedure SetCBreak(breakvalue: Boolean)`

Visibility: default

Description: `SetCBreak` sets the status of CTRL-Break checking. When `BreakValue` is `false`, then dos only checks for the CTRL-Break key-press when I/O is performed. When it is set to `True`, then a check is done at every system call.

Remark: Under non-dos and non-Amiga operating systems, this call does nothing.

Errors: None.

See also: `GetCBreak` ([435](#))

SetDate

Synopsis: Set system date

Declaration: `procedure SetDate(year: Word;month: Word;day: Word)`

Visibility: default

Description: `SetDate` sets the system's internal date. Year is a number between 1980 and 2099.

Remark: On a unix machine, there must be root privileges, otherwise this routine will do nothing. On other unix systems, this call currently does nothing.

Errors: None.

See also: `GetDate` ([435](#)), `SetTime` ([444](#))

SetFAttr

Synopsis: Set file attributes

Declaration: `procedure SetFAttr(var f; attr: Word)`

Visibility: default

Description: `SetFAttr` sets the file attributes of the file-variable `F`. `F` can be a untyped or typed file, or of type `Text`. `F` must have been assigned, but not opened. The attributes can be a sum of the following constants:

- `ReadOnly`
- `Hidden`
- `SysFile`
- `VolumeId`
- `Directory`
- `Archive`

Remark: Under unix like systems (such as linux and BeOS) the call exists, but is not implemented, i.e. it does nothing.

Errors: Errors are reported in `DosError`.

See also: `GetFAttr` ([436](#))

SetFTime

Synopsis: Set file modification time.

Declaration: `procedure SetFTime(var f; time: LongInt)`

Visibility: default

Description: `SetFTime` sets the modification time of a file, this time is encoded and must be encoded with `PackTime`. `F` must be a file type, which has been assigned, and opened.

Remark: Under unix like systems (such as linux and BeOS) the call exists, but is not implemented, i.e. it does nothing.

Errors: Errors are reported in `DosError`

See also: `GetFTime` ([437](#)), `PackTime` ([441](#)), `UnPackTime` ([445](#))

SetIntVec

Synopsis: Set interrupt vector

Declaration: `procedure SetIntVec(intno: Byte; vector: pointer)`

Visibility: default

Description: `SetIntVec` sets interrupt vector `IntNo` to `Vector`. `Vector` should point to an interrupt procedure.

Remark: This call does nothing, it is present for compatibility only.

Errors: None.

See also: `GetIntVec` ([438](#))

SetTime

Synopsis: Set system time

Declaration: `procedure SetTime(hour: Word; minute: Word; second: Word; sec100: Word)`

Visibility: default

Description: `SetTime` sets the system's internal clock. The `Hour` parameter is on a 24-hour time scale.

Remark: On a linux machine, there must be root privileges, otherwise this routine will do nothing. On other unix systems, this call currently does nothing.

Errors: None.

See also: `GetTime` ([439](#)), `SetDate` ([442](#))

SetVerify

Synopsis: Set verify flag

Declaration: `procedure SetVerify(verify: Boolean)`

Visibility: default

Description: `SetVerify` sets the status of the verify flag under dos. When `Verify` is `True`, then dos checks data which are written to disk, by reading them after writing. If `Verify` is `False`, then data written to disk are not verified.

Remark: Under non-dos operating systems (excluding os/2 applications running under vanilla dos), `Verify` is always `True`.

Errors: None.

See also: `SetVerify` ([444](#))

SwapVectors

Synopsis: Swap interrupt vectors

Declaration: `procedure SwapVectors`

Visibility: default

Description: `SwapVectors` swaps the contents of the internal table of interrupt vectors with the current contents of the interrupt vectors. This is called typically in before and after an `Exec` call.

Remark: Under certain operating systems, this routine may be implemented as an empty stub.

Errors: None.

See also: `Exec` ([432](#)), `SetIntVec` ([443](#))

UnixDateToDt

Synopsis: Convert a unix timestamp to a DateTime record

Declaration: `procedure UnixDateToDt(SecsPast: LongInt; var Dt: DateTime)`

Visibility: default

Description: `DTToUnixDate` converts the unix timestamp value in `SecsPast` to a `DateTime` representation in DT. It is an internal function, implemented on Unix platforms, and should not be used.

Errors: None.

See also: `DTToUnixDate` (430), `PackTime` (441), `UnpackTime` (445), `GetTime` (439), `SetTime` (444)

UnpackTime

Synopsis: Unpack packed file time to a DateTime value

Declaration: `procedure UnpackTime(p: LongInt; var t: DateTime)`

Visibility: default

Description: `UnPackTime` converts the file-modification time in `p` to a `DateTime` record. The file-modification time can be returned by `GetFTime`, `FindFirst` or `FindNext` calls.

For an example, see `PackTime` (441).

Errors: None.

See also: `GetFTime` (437), `FindFirst` (433), `FindNext` (434), `PackTime` (441)

weekday

Synopsis: Return the day of the week

Declaration: `function weekday(y: LongInt; m: LongInt; d: LongInt) : LongInt`

Visibility: default

Description: `WeekDay` returns the day of the week on which the day Y/M/D falls. Sunday is represented by 0, Saturday is 6.

Errors: On error, -1 is returned.

See also: `PackTime` (441), `UnpackTime` (445), `GetTime` (439), `SetTime` (444)

Chapter 6

Reference for unit 'dxeload'

6.1 Overview

The `dxeload` unit was implemented by Pierre Mueller for dos, it allows to load a DXE file (an object file with 1 entry point) into memory and return a pointer to the entry point.

It exists only for dos.

6.2 Procedures and functions

dxeload

Synopsis: Load DXE file in memory

Declaration: `function dxeload(filename: String) : pointer`

Visibility: default

Description: `dxeload` loads the contents of the file `filename` into memory. It performs the necessary relocations in the object code, and returns then a pointer to the entry point of the code.

For an example, see the `emu387` ([450](#)) unit in the RTL.

Errors: If an error occurs during the load or relocations, `Nil` is returned.

Chapter 7

Reference for unit 'dynlibs'

7.1 Overview

The Dynlibs unit provides support for dynamically loading shared libraries. It is available only on those platforms that support shared libraries. The functionality available here may only be a part of the functionality available on each separate platform, in the interest of portability.

On unix platforms, using this unit will cause the program to be linked to the C library, as most shared libraries are implemented in C and the dynamical linker too.

7.2 Constants, types and variables

Constants

`NilHandle = nil`

Correctly typed Nil handle - returned on error by `LoadLibrary` ([448](#))

Types

`TLibHandle = Pointer`

`TLibHandle` should be considered an opaque type. It is defined differently on various platforms. The definition shown here depends on the platform for which the documentation was generated.

7.3 Procedures and functions

FreeLibrary

Synopsis: For compatibility with Delphi/Windows: Unload a library

Declaration: `function FreeLibrary(Lib: TLibHandle) : Boolean`

Visibility: default

Description: `FreeLibrary` provides the same functionality as `UnloadLibrary` ([448](#)), and is provided for compatibility with Delphi.

See also: `UnloadLibrary` ([448](#))

GetProcAddress

Synopsis: For compatibility with Delphi/Windows: Get the address of a procedure

Declaration: `function GetProcAddress(Lib: TLibHandle; ProcName: AnsiString) : Pointer`

Visibility: default

Description: `GetProcAddress` provides the same functionality as `GetProcedureAddress` (448), and is provided for compatibility with Delphi.

See also: `GetProcedureAddress` (448)

GetProcedureAddress

Synopsis: Get the address of a procedure or symbol in a dynamic library.

Declaration: `function GetProcedureAddress(Lib: TLibHandle; ProcName: AnsiString)
: Pointer`

Visibility: default

Description: `GetProcedureAddress` returns a pointer to the location in memory of the symbol `ProcName` in the dynamically loaded library specified by its handle `lib`. If the symbol cannot be found or the handle is invalid, `Nil` is returned.

On Windows, only an exported procedure or function can be searched this way. On Unix platforms the location of any exported symbol can be retrieved this way.

Errors: If the symbol cannot be found, `Nil` is returned.

See also: `LoadLibrary` (448), `UnLoadLibrary` (448)

LoadLibrary

Synopsis: Load a dynamic library and return a handle to it.

Declaration: `function LoadLibrary(Name: AnsiString) : TLibHandle`

Visibility: default

Description: `LoadLibrary` loads a dynamic library in file `Name` and returns a handle to it. If the library cannot be loaded, `NilHandle` (447) is returned.

No assumptions should be made about the location of the loaded library if a relative pathname is specified. The behaviour is dependent on the platform. Therefore it is best to specify an absolute pathname if possible.

Errors: On error, `NilHandle` (447) is returned.

See also: `UnLoadLibrary` (448), `GetProcedureAddress` (448)

UnLoadLibrary

Synopsis: Unload a previously loaded library

Declaration: `function UnloadLibrary(Lib: TLibHandle) : Boolean`

Visibility: default

Description: `UnloadLibrary` unloads a previously loaded library (specified by the handle `lib`). The call returns `True` if succesful, `False` otherwisa.

Errors: On error, `False` is returned.

See also: `LoadLibrary` ([448](#)), `GetProcAddress` ([448](#))

Chapter 8

Reference for unit 'emu387'

8.1 Overview

The `emu387` unit was written by Pierre Mueller for dos. It sets up the coprocessor emulation for FPC under dos. It is not necessary to use this unit on other OS platforms because they either simply do not run on a machine without coprocessor, or they provide the coprocessor emulation themselves.

It shouldn't be necessary to use the function in this unit, it should be enough to place this unit in the `uses` clause of your program to enable the coprocessor emulation under dos. The unit initialization code will try and load the coprocessor emulation code and initialize it.

8.2 Procedures and functions

npxsetup

Synopsis: Set up coprocessor emulation.

Declaration: `procedure npxsetup(prog_name: String)`

Visibility: `default`

Description: `npxsetup` checks whether a coprocessor is found. If not, it loads the file `wmemu387.dxe` into memory and initializes the code in it.

If the environment variable `387` is set to `N`, then the emulation will be loaded, even if there is a coprocessor present. If the variable doesn't exist, or is set to any other value, the unit will try to detect the presence of a coprocessor unit.

The function searches the file `wmemu387.dxe` in the following way:

- 1.If the environment variable `EMU387` is set, then it is assumed to point at the `wmemu387.dxe` file.
- 2.if the environment variable `EMU387` does not exist, then the function will take the path part of `prog_name` and look in that directory for the file `wmemu387.dxe`.

It should never be necessary to call this function, because the initialization code of the unit contains a call to the function with as an argument `paramstr(0)`. This means that you should deliver the file `wmemu387.dxe` together with your program.

Errors: If there is an error, an error message is printed to standard error, and the program is halted, since any floating-point code is bound to fail anyhow.

Chapter 9

Reference for unit 'getopts'

9.1 Overview

This document describes the GETOPTS unit for Free Pascal. It was written for linux by Michael Van Canneyt. It now also works for all supported platforms.

The getopts unit provides a mechanism to handle command-line options in a structured way, much like the GNU getopts mechanism. It allows you to define the valid options for your program, and the unit will then parse the command-line options for you, and inform you of any errors.

9.2 Constants, types and variables

Constants

`EndOfOptions = #255`

Returned by `getopt` ([453](#)), `getlongopts` ([452](#)) to indicate that there are no more options.

`No_Argument = 0`

Specifies that a long option does not take an argument.

`Optional_Argument = 2`

Specifies that a long option optionally takes an argument.

`OptSpecifier : Set of Char = ['-']`

Character indicating an option on the command-line.

`Required_Argument = 1`

Specifies that a long option needs an argument.

Types

`Orderings = (require_order, permute, return_in_order)`

Table 9.1: Enumeration values for type Orderings

Value	Explanation
permute	Change command-line options.
require_order	Don't touch the ordering of the command-line options
return_in_order	Return options in the correct order.

Command-line ordering options.

POption = ^TOption

Pointer to TOption (452) record.

```
TOption = record
  Name : String;
  Has_arg : Integer;
  Flag : PChar;
  Value : Char;
end
```

The TOption type is used to communicate the long options to GetLongOpts (452). The Name field is the name of the option. Has_arg specifies if the option wants an argument, Flag is a pointer to a char, which is set to Value, if it is non-nil.

Variables

OptArg : String

Set to the argument of an option, if the option needs one.

OptErr : Boolean

Indicates whether getopt () prints error messages.

OptInd : LongInt

when all options have been processed, optind is the index of the first non-option parameter. This is a read-only variable. Note that it can become equal to paramcount+1.

OptOpt : Char

In case of an error, contains the character causing the error.

9.3 Procedures and functions

GetLongOpts

Synopsis: Return next long option.

Declaration: function GetLongOpts(ShortOpts: String; LongOpts: POption;
var Longind: LongInt) : Char

Visibility: default

Description: Returns the next option found on the command-line, taking into account long options as well. If no more options are found, returns `EndOfOptions`. If the option requires an argument, it is returned in the `OptArg` variable.

`ShortOptions` is a string containing all possible one-letter options. (see [Getopt \(453\)](#) for its description and use) `LongOpts` is a pointer to the first element of an array of `Option` records, the last of which needs a name of zero length.

The function tries to match the names even partially (i.e. `-app` will match e.g. the `append` option), but will report an error in case of ambiguity. If the option needs an argument, set `Has_arg` to `Required_argument`, if the option optionally has an argument, set `Has_arg` to `Optional_argument`. If the option needs no argument, set `Has_arg` to zero.

Required arguments can be specified in two ways :

1. Pasted to the option : `-option=value`
2. As a separate argument : `-option value`

Optional arguments can only be specified through the first method.

Errors: see [Getopt \(453\)](#).

See also: [Getopt \(453\)](#)

GetOpt

Synopsis: Get next short option.

Declaration: `function GetOpt(ShortOpts: String) : Char`

Visibility: default

Description: Returns the next option found on the command-line. If no more options are found, returns `EndOfOptions`. If the option requires an argument, it is returned in the `OptArg` variable.

`ShortOptions` is a string containing all possible one-letter options. If a letter is followed by a colon (:), then that option needs an argument. If a letter is followed by 2 colons, the option has an optional argument. If the first character of `shortoptions` is a '+' then options following a non-option are regarded as non-options (standard Unix behavior). If it is a '-', then all non-options are treated as arguments of a option with character #0. This is useful for applications that require their options in the exact order as they appear on the command-line. If the first character of `shortoptions` is none of the above, options and non-options are permuted, so all non-options are behind all options. This allows options and non-options to be in random order on the command line.

Errors: Errors are reported through giving back a '?' character. `OptOpt` then gives the character which caused the error. If `OptErr` is `True` then `getopt` prints an error-message to `stdout`.

See also: [GetLongOpts \(452\)](#)

Listing: `./optex/optex.pp`

```
program testopt;

{ Program to depmonstrate the getopt function. }

{
  Valid calls to this program are
  optex --verbose --add me--delete you
```

```

    optex --append --create child
    optex -ab -c me -d you
    and so on
}
uses getopt;

var c : char;
    optionindex : Longint;
    theopts : array[1..7] of TOption;

begin
    with theopts[1] do
        begin
            name := 'add';
            has_arg := 1;
            flag := nil;
            value := #0;
        end;
    with theopts[2] do
        begin
            name := 'append';
            has_arg := 0;
            flag := nil;
            value := #0;
        end;
    with theopts[3] do
        begin
            name := 'delete';
            has_arg := 1;
            flag := nil;
            value := #0;
        end;
    with theopts[4] do
        begin
            name := 'verbose';
            has_arg := 0;
            flag := nil;
            value := #0;
        end;
    with theopts[5] do
        begin
            name := 'create';
            has_arg := 1;
            flag := nil;
            value := 'c';
        end;
    with theopts[6] do
        begin
            name := 'file';
            has_arg := 1;
            flag := nil;
            value := #0;
        end;
    with theopts[7] do
        begin
            name := '';
            has_arg := 0;
            flag := nil;

```

```
end;
c:=#0;
repeat
  c:=getlongopts('abc:d:012',@theopts[1],optionindex);
  case c of
    '1','2','3','4','5','6','7','8','9' :
      begin
        writeln('Got optind : ',c)
      end;
    #0 : begin
        write('Long option : ',theopts[optionindex].name);
        if theopts[optionindex].has_arg>0 then
          writeln(' With value : ',optarg)
        else
          writeln
        end;
        'a' : writeln('Option a. ');
        'b' : writeln('Option b. ');
        'c' : writeln('Option c : ',optarg);
        'd' : writeln('Option d : ',optarg);
        '?' : writeln('Error with opt : ',optopt);
      end; { case }
  until c=endofoptions;
if optind<=paramcount then
  begin
    write('Non options : ');
    while optind<=paramcount do
      begin
        write(paramstr(optind),' ');
        inc(optind)
      end;
    writeln
  end
end.
```

Chapter 10

Reference for unit 'go32'

10.1 Real mode callbacks

The callback mechanism can be thought of as the converse of calling a real mode procedure (i.e. interrupt), which allows your program to pass information to a real mode program, or obtain services from it in a manner that's transparent to the real mode program. In order to make a real mode callback available, you must first get the real mode callback address of your procedure and the selector and offset of a register data structure. This real mode callback address (this is a segment:offset address) can be passed to a real mode program via a software interrupt, a dos memory block or any other convenient mechanism. When the real mode program calls the callback (via a far call), the DPMI host saves the registers contents in the supplied register data structure, switches into protected mode, and enters the callback routine with the following settings:

- interrupts disabled
- `%CS : %EIP` = 48 bit pointer specified in the original call to `get_rm_callback` (473)
- `%DS : %ESI` = 48 bit pointer to to real mode `SS : SP`
- `%ES : %EDI` = 48 bit pointer of real mode register data structure.
- `%SS : %ESP` = locked protected mode stack
- All other registers undefined

The callback procedure can then extract its parameters from the real mode register data structure and/or copy parameters from the real mode stack to the protected mode stack. Recall that the segment register fields of the real mode register data structure contain segment or paragraph addresses that are not valid in protected mode. Far pointers passed in the real mode register data structure must be translated to virtual addresses before they can be used with a protected mode program. The callback procedure exits by executing an `IRET` with the address of the real mode register data structure in `%ES : %EDI`, passing information back to the real mode caller by modifying the contents of the real mode register data structure and/or manipulating the contents of the real mode stack. The callback procedure is responsible for setting the proper address for resumption of real mode execution into the real mode register data structure; typically, this is accomplished by extracting the return address from the real mode stack and placing it into the `%CS : %EIP` fields of the real mode register data structure. After the `IRET`, the DPMI host switches the CPU back into real mode, loads ALL registers with the contents of the real mode register data structure, and finally returns control to the real mode program. All variables and code touched by the callback procedure **MUST** be locked to prevent page faults.

See also: `get_rm_callback` (473), `free_rm_callback` (469), `lock_code` (481), `lock_data` (482)

10.2 Executing software interrupts

Simply execute a `realintr()` call with the desired interrupt number and the supplied register data structure. But some of these interrupts require you to supply them a pointer to a buffer where they can store data to or obtain data from in memory. These interrupts are real mode functions and so they only can access the first Mb of linear address space, not FPC's data segment. For this reason FPC supplies a pre-initialized dos memory location within the GO32 unit. This buffer is internally used for dos functions too and so it's contents may change when calling other procedures. It's size can be obtained with `tb_size` (490) and it's linear address via `transfer_buffer` (490). Another way is to allocate a completely new dos memory area via the `global_dos_alloc` (478) function for your use and supply its real mode address.

See also: `tb_size` (490), `transfer_buffer` (490), `global_dos_alloc` (478), `global_dos_free` (480), `realintr` (484)

10.3 Software interrupts

Ordinarily, a handler installed with `set_pm_interrupt` (487) only services software interrupts that are executed in protected mode; real mode software interrupts can be redirected by `set_rm_interrupt` (488).

See also: `set_rm_interrupt` (488), `get_rm_interrupt` (476), `set_pm_interrupt` (487), `get_pm_interrupt` (473), `lock_data` (482), `lock_code` (481), `enable` (468), `disable` (467), `outportb` (483)

10.4 Hardware interrupts

Hardware interrupts are generated by hardware devices when something unusual happens; this could be a keypress or a mouse move or any other action. This is done to minimize CPU time, else the CPU would have to check all installed hardware for data in a big loop (this method is called 'polling') and this would take much time. A standard IBM-PC has two interrupt controllers, that are responsible for these hardware interrupts: both allow up to 8 different interrupt sources (IRQs, interrupt requests). The second controller is connected to the first through IRQ 2 for compatibility reasons, e.g. if controller 1 gets an IRQ 2, he hands the IRQ over to controller 2. Because of this up to 15 different hardware interrupt sources can be handled. IRQ 0 through IRQ 7 are mapped to interrupts 8h to Fh and the second controller (IRQ 8 to 15) is mapped to interrupt 70h to 77h. All of the code and data touched by these handlers **MUST** be locked (via the various locking functions) to avoid page faults at interrupt time. Because hardware interrupts are called (as in real mode) with interrupts disabled, the handler has to enable them before it returns to normal program execution. Additionally a hardware interrupt must send an EOI (end of interrupt) command to the responsible controller; this is accomplished by sending the value 20h to port 20h (for the first controller) or A0h (for the second controller). The following example shows how to redirect the keyboard interrupt.

10.5 Disabling interrupts

The GO32 unit provides the two procedures `disable()` and `enable()` to disable and enable all interrupts.

10.6 Creating your own interrupt handlers

Interrupt redirection with FPC pascal is done via the `set_pm_interrupt()` for protected mode interrupts or via the `set_rm_interrupt()` for real mode interrupts.

10.7 Protected mode interrupts vs. Real mode interrupts

As mentioned before, there's a distinction between real mode interrupts and protected mode interrupts; the latter are protected mode programs, while the former must be real mode programs. To call a protected mode interrupt handler, an assembly 'int' call must be issued, while the other is called via the `realintr()` or `intr()` function. Consequently, a real mode interrupt then must either reside in dos memory (<1MB) or the application must allocate a real mode callback address via the `get_rm_callback()` function.

10.8 Handling interrupts with DPMI

The interrupt functions are real-mode procedures; they normally can't be called in protected mode without the risk of an protection fault. So the DPMI host creates an interrupt descriptor table for the application. Initially all software interrupts (except for int 31h, 2Fh and 21h function 4Ch) or external hardware interrupts are simply directed to a handler that reflects the interrupt in real-mode, i.e. the DPMI host's default handlers switch the CPU to real-mode, issue the interrupt and switch back to protected mode. The contents of general registers and flags are passed to the real mode handler and the modified registers and flags are returned to the protected mode handler. Segment registers and stack pointer are not passed between modes.

10.9 Interrupt redirection

Interrupts are program interruption requests, which in one or another way get to the processor; there's a distinction between software and hardware interrupts. The former are explicitly called by an 'int' instruction and are a bit comparable to normal functions. Hardware interrupts come from external devices like the keyboard or mouse. Functions that handle hardware interrupts are called handlers.

10.10 Processor access

These are some functions to access various segment registers (%cs, %ds, %ss) which makes your work a bit easier.

See also: `get_cs` (469), `get_ds` (470), `get_ss` (478)

10.11 I/O port access

The I/O port access is done via the various `inportb` (480), `outportb` (483) functions which are available. Additionally Free Pascal supports the Turbo Pascal `PORT[]`-arrays but it is by no means recommended to use them, because they're only for compatibility purposes.

See also: `outportb` (483), `inportb` (480)

10.12 dos memory access

Dos memory is accessed by the predefined `dosmemselector` selector; the GO32 unit additionally provides some functions to help you with standard tasks, like copying memory from heap to dos memory and the likes. Because of this it is strongly recommended to use them, but you are still free

to use the provided standard memory accessing functions which use 48 bit pointers. The third, but only thought for compatibility purposes, is using the `mem[]`-arrays. These arrays map the whole 1 Mb dos space. They shouldn't be used within new programs. To convert a segment:offset real mode address to a protected mode linear address you have to multiply the segment by 16 and add its offset. This linear address can be used in combination with the `DOSMEMSELECTOR` variable.

See also: `dosmemget` (461), `dosmemput` (461), `dosmemmove` (461), `dosmemfillchar` (460), `dosmemfillword` (460), `seg_move` (486), `seg_fillchar` (485), `seg_fillword` (486)

10.13 FPC specialities

The `%ds` and `%es` selector MUST always contain the same value or some system routines may crash when called. The `%fs` selector is preloaded with the `DOSMEMSELECTOR` variable at startup, and it MUST be restored after use, because again FPC relies on this for some functions. Luckily we asm programmers can still use the `%gs` selector for our own purposes, but for how long ?

See also: `get_cs` (469), `get_ds` (470), `get_ss` (478), `allocate_ldt_descriptors` (463), `free_ldt_descriptor` (468), `segment_to_descriptor` (485), `get_next_selector_increment_value` (472), `get_segment_base_address` (477), `set_segment_base_address` (489), `set_segment_limit` (489), `create_code_segment_alias_descriptor` (467)

10.14 Selectors and descriptors

Descriptors are a bit like real mode segments; they describe (as the name implies) a memory area in protected mode. A descriptor contains information about segment length, its base address and the attributes of it (i.e. type, access rights, ...). These descriptors are stored internally in a so-called descriptor table, which is basically an array of such descriptors. Selectors are roughly an index into this table. Because these 'segments' can be up to 4 GB in size, 32 bits aren't sufficient anymore to describe a single memory location like in real mode. 48 bits are now needed to do this, a 32 bit address and a 16 bit sized selector. The GO32 unit provides the `tseginfo` record to store such a pointer. But due to the fact that most of the time data is stored and accessed in the `%ds` selector, FPC assumes that all pointers point to a memory location of this selector. So a single pointer is still only 32 bits in size. This value represents the offset from the data segment base address to this memory location.

10.15 What is DPMI

The dos Protected Mode Interface helps you with various aspects of protected mode programming. These are roughly divided into descriptor handling, access to dos memory, management of interrupts and exceptions, calls to real mode functions and other stuff. Additionally it automatically provides swapping to disk for memory intensive applications. A DPMI host (either a Windows dos box or `CWSDPMI.EXE`) provides these functions for your programs.

10.16 Overview

This document describes the GO32 unit for the Free Pascal compiler under dos. It was donated by Thomas Schatzl (`tom_at_work@geocities.com`), for which my thanks. This unit was first written for dos by Florian Klaempfl.

Only the GO32V2 DPMI mode is discussed by me here due to the fact that new applications shouldn't be created with the older GO32V1 model. The go32v2 version is much more advanced and better. Additionally a lot of functions only work in DPMI mode anyway. I hope the following explanations and introductions aren't too confusing at all. If you notice an error or bug send it to the FPC mailing list or directly to me. So let's get started and happy and error free coding I wish you.... Thomas Schatzl, 25. August 1998

10.17 Constants, types and variables

Constants

`auxcarryflag = $010`

Check for auxiliary carry flag in `trealregs` ([463](#))

`carryflag = $001`

Check for carry flag in `trealregs` ([463](#))

`directionflag = $400`

Check for direction flag in `trealregs` ([463](#))

`dosmemfillchar : procedure(seg: Word;ofs: Word;count: LongInt;c: Char) = @dpmi_dosmemfillchar`

Sets a region of dos memory to a specific byte value.

Parameters:

seg real mode segment.

ofs real mode offset.

count number of bytes to set.

c value to set memory to.

Notes: No range check is performed.

`dosmemfillword : procedure(seg: Word;ofs: Word;count: LongInt;w: Word) = @dpmi_dosmemfillword`

Sets a region of dos memory to a specific word value.

Parameters:

seg real mode segment.

ofs real mode offset.

count number of words to set.

w value to set memory to.

Notes: No range check is performed.

`dosmemget : procedure(seg: Word;ofs: Word;var data;count: LongInt) = @dpmi_dosmemget`

Copies data from the dos memory onto the heap.

Parameters:

seg source real mode segment.

ofs source real mode offset.

data destination.

count number of bytes to copy.

Notes: No range checking is performed.

For an example, see `global_dos_alloc` ([478](#)).

```
dosmemmove : procedure(sseg: Word;sofs: Word;dseg: Word;dofs: Word;count: LongInt) =
```

Copies count bytes of data between two dos real mode memory locations.

Parameters:

sseg source real mode segment.

sofs source real mode offset.

dseg destination real mode segment.

dofs destination real mode offset.

count number of bytes to copy.

Notes: No range check is performed in any way.

```
dosmemput : procedure(seg: Word;ofs: Word;var data;count: LongInt) = @dpmi_dosmemput
```

Copies heap data to dos real mode memory.

Parameters:

seg destination real mode segment.

ofs destination real mode offset.

data source.

count number of bytes to copy.

Notes: No range checking is performed.

For an example, see `global_dos_alloc` ([478](#)).

```
interruptflag = $200
```

Check for interrupt flag in `trealregs` ([463](#))

```
overflowflag = $800
```

Check for overflow flag in `trealregs` ([463](#))

```
parityflag = $004
```

Check for parity flag in trealregs ([463](#))

```
rm_dpml = 4
```

get_run_mode ([476](#)) return value: DPMI (e.g. dos box or 386Max)

```
rm_raw = 1
```

get_run_mode ([476](#)) return value: raw (without HIMEM)

```
rm_unknown = 0
```

get_run_mode ([476](#)) return value: Unknown runmode

```
rm_vcpi = 3
```

get_run_mode ([476](#)) return value: VCPI (with HIMEM and EMM386)

```
rm_xms = 2
```

get_run_mode ([476](#)) return value: XMS (with HIMEM, without EMM386)

```
signflag = $080
```

Check for sign flag in trealregs ([463](#))

```
trapflag = $100
```

Check for trap flag in trealregs ([463](#))

```
zeroflag = $040
```

Check for zero flag in trealregs ([463](#))

Types

```
registers = trealregs
```

Alias for trealregs ([463](#))

```
tmeminfo = record
  available_memory : LongInt;
  available_pages : LongInt;
  available_lockable_pages : LongInt;
  linear_space : LongInt;
  unlocked_pages : LongInt;
  available_physical_pages : LongInt;
  total_physical_pages : LongInt;
  free_linear_space : LongInt;
  max_pages_in_paging_file : LongInt;
  reserved0 : LongInt;
  reserved1 : LongInt;
  reserved2 : LongInt;
end
```

`tmeminfo` Holds information about the memory allocation, etc.

NOTE: The value of a field is -1 (0ffffffh) if the value is unknown, it's only guaranteed, that `available_memory` contains a valid value. The size of the pages can be determined by the `get_page_size()` function.

```
trealregs = record
end
```

The `trealregs` type contains the data structure to pass register values to a interrupt handler or real mode callback.

```
tseginfo = record
  offset : pointer;
  segment : Word;
end
```

This record is used to store a full 48-bit pointer. This may be either a protected mode selector:offset address or in real mode a segment:offset address, depending on application.

See also: Selectors and descriptors, dos memory access, Interrupt redirection

Variables

```
dosmemselector : Word
```

Selector to the dos memory. The whole dos memory is automatically mapped to this single descriptor at startup. This selector is the recommended way to access dos memory.

```
int31error : Word
```

This variable holds the result of a DPMI interrupt call. Any nonzero value must be treated as a critical failure.

10.18 Procedures and functions

allocate_ldt_descriptors

Synopsis: Allocate a number of descriptors

Declaration: `function allocate_ldt_descriptors(count: Word) : Word`

Visibility: default

Description: Allocates a number of new descriptors.

Parameters:

count: \ specifies the number of requested unique descriptors.

Return value: The base selector.

Remark: Notes: The descriptors allocated must be initialized by the application with other function calls. This function returns descriptors with a limit and size value set to zero. If more than one descriptor

was requested, the function returns a base selector referencing the first of a contiguous array of descriptors. The selector values for subsequent descriptors in the array can be calculated by adding the value returned by the `get_next_selector_increment_value` (472) function.

Errors: Check the `int31error` (463) variable.

See also: `free_ldt_descriptor` (468), `get_next_selector_increment_value` (472), `segment_to_descriptor` (485), `create_code_segment_alias_descriptor` (467), `set_segment_limit` (489), `set_segment_base_address` (489)

Listing: `./go32ex/seldes.pp`

```
{ $mode delphi }
uses
    crt ,
    go32;

const
    maxx = 80;
    maxy = 25;
    bytespercell = 2;
    screensize = maxx * maxy * bytespercell;

    linB8000 = $B800 * 16;

type
    string80 = string[80];

var
    text_save : array[0..screensize-1] of byte;
    text_oldx , text_oldy : Word;

    text_sel : Word;

procedure status(s : string80);
begin
    gotoxy(1, 1); clreol; write(s); readkey;
end;

procedure selinfo(sel : Word);
begin
    gotoxy(1, 24);
    clreol; writeln('Descriptor base address : $',
        hexstr(get_segment_base_address(sel), 8));
    clreol; write('Descriptor limit : ', get_segment_limit(sel));
end;

function makechar(ch : char; color : byte) : Word;
begin
    result := byte(ch) or (color shl 8);
end;

begin
    seg_move(dosmemselector, linB8000, get_ds, longint(@text_save),
        screensize);
    text_oldx := wherex; text_oldy := wherey;
    seg_fillword(dosmemselector, linB8000, screensize div 2,
        makechar(' ', Black or (Black shl 4)));
    status('Creating selector ''text_sel'' to a part of ' +
```

```

    'text screen memory');
text_sel := allocate_ldt_descriptors(1);
set_segment_base_address(text_sel,
    linB8000 + bytespercell * maxx * 1);
set_segment_limit(text_sel, screensize - 1 - bytespercell *
    maxx * 3);
selinfo(text_sel);

status('and clearing entire memory selected by ''text_sel'' +
    ' descriptor');
seg_fillword(text_sel, 0, (get_segment_limit(text_sel)+1) div 2,
    makechar(' ', LightBlue shl 4));

status('Notice that only the memory described by the' +
    ' descriptor changed, nothing else');

status('Now reducing it''s limit and base and setting it''s ' +
    'described memory');
set_segment_base_address(text_sel,
    get_segment_base_address(text_sel) + bytespercell * maxx);
set_segment_limit(text_sel,
    get_segment_limit(text_sel) - bytespercell * maxx * 2);
selinfo(text_sel);
status('Notice that the base addr increased by one line but ' +
    'the limit decreased by 2 lines');
status('This should give you the hint that the limit is ' +
    'relative to the base');
seg_fillword(text_sel, 0, (get_segment_limit(text_sel)+1) div 2,
    makechar(#176, LightMagenta or Brown shl 4));

status('Now let''s get crazy and copy 10 lines of data from ' +
    'the previously saved screen');
seg_move(get_ds, longint(@text_save), text_sel,
    maxx * bytespercell * 2, maxx * bytespercell * 10);

status('At last freeing the descriptor and restoring the old ' +
    ' screen contents..');
status('I hope this little program may give you some hints on ' +
    'working with descriptors');
free_ldt_descriptor(text_sel);
seg_move(get_ds, longint(@text_save), dosmemselector,
    linB8000, screensize);
gotoxy(text_oldx, text_oldy);
end.

```

allocate_memory_block

Synopsis: Allocate a block of linear memory

Declaration: function allocate_memory_block(size: LongInt) : LongInt

Visibility: default

Description: Allocates a block of linear memory.

Parameters:

size:Size of requested linear memory block in bytes.

Returned values: blockhandle - the memory handle to this memory block. Linear address of the requested memory.

Remark: *warning* According to my DPMI docs this function is not implemented correctly. Normally you should also get a blockhandle to this block after successful operation. This handle can then be used to free the memory block afterwards or use this handle for other purposes. Since the function isn't implemented correctly, and doesn't return a blockhandle, the block can't be deallocated and is hence unusable ! This function doesn't allocate any descriptors for this block, it's the applications responsibility to allocate and initialize for accessing this memory.

Errors: Check the int31error (463) variable.

See also: free_memory_block (469)

copyfromdos

Synopsis: Copy data from DOS to to heap

Declaration: `procedure copyfromdos(var addr;len: LongInt)`

Visibility: default

Description: Copies data from the pre-allocated dos memory transfer buffer to the heap.

Parameters:

addrdata to copy to.

lennumber of bytes to copy to heap.

Notes: Can only be used in conjunction with the dos memory transfer buffer.

Errors: Check the int31error (463) variable.

See also: tb_size (490), transfer_buffer (490), copytodos (466)

copytodos

Synopsis: Copy data from heap to DOS memory

Declaration: `procedure copytodos(var addr;len: LongInt)`

Visibility: default

Description: Copies data from heap to the pre-allocated dos memory buffer.

Parameters:

addrdata to copy from.

lennumber of bytes to copy to dos memory buffer.

Notes: This function fails if you try to copy more bytes than the transfer buffer is in size. It can only be used in conjunction with the transfer buffer.

Errors: Check the int31error (463) variable.

See also: tb_size (490), transfer_buffer (490), copyfromdos (466)

create_code_segment_alias_descriptor

Synopsis: Create new descriptor from existing descriptor

Declaration: `function create_code_segment_alias_descriptor(seg: Word) : Word`

Visibility: default

Description: Creates a new descriptor that has the same base and limit as the specified descriptor.

Parameters:

segDescriptor.

Return values: The data selector (alias).

Notes: In effect, the function returns a copy of the descriptor. The descriptor alias returned by this function will not track changes to the original descriptor. In other words, if an alias is created with this function, and the base or limit of the original segment is then changed, the two descriptors will no longer map the same memory.

Errors: Check the `int31error` ([463](#)) variable.

See also: `allocate_ldt_descriptors` ([463](#)), `set_segment_limit` ([489](#)), `set_segment_base_address` ([489](#))

disable

Synopsis: Disable hardware interrupts

Declaration: `procedure disable`

Visibility: default

Description: Disables all hardware interrupts by execution a CLI instruction.

Errors: None.

See also: `enable` ([468](#))

dpmi_dosmemfillchar

Declaration: `procedure dpmi_dosmemfillchar(seg: Word;ofs: Word;count: LongInt;
c: Char)`

Visibility: default

dpmi_dosmemfillword

Declaration: `procedure dpmi_dosmemfillword(seg: Word;ofs: Word;count: LongInt;
w: Word)`

Visibility: default

dpmi_dosmemget

Declaration: `procedure dpmi_dosmemget(seg: Word;ofs: Word;var data;count: LongInt)`

Visibility: default

dpmi_dosmemmove

Declaration: `procedure dpmi_dosmemmove(sseg: Word;sofs: Word;dseg: Word;dofs: Word;count: LongInt)`

Visibility: default

dpmi_dosmemput

Declaration: `procedure dpmi_dosmemput(seg: Word;ofs: Word;var data;count: LongInt)`

Visibility: default

enable

Synopsis: Enable hardware interrupts

Declaration: `procedure enable`

Visibility: default

Description: Enables all hardware interrupts by executing a STI instruction.

Errors: None.

See also: `disable` ([467](#))

free_ldt_descriptor

Synopsis: Free a descriptor

Declaration: `function free_ldt_descriptor(d: Word) : Boolean`

Visibility: default

Description: Frees a previously allocated descriptor.

Parameters:

des The descriptor to be freed.

Return value: `True` if successful, `False` otherwise. Notes: After this call this selector is invalid and must not be used for any memory operations anymore. Each descriptor allocated with `allocate_ldt_descriptors` ([463](#)) must be freed individually with this function, even if it was previously allocated as a part of a contiguous array of descriptors.

For an example, see `allocate_ldt_descriptors` ([463](#)).

Errors: Check the `int31error` ([463](#)) variable.

See also: `allocate_ldt_descriptors` ([463](#)), `get_next_selector_increment_value` ([472](#))

free_memory_block

Synopsis: Free allocated memory block

Declaration: `function free_memory_block(blockhandle: LongInt) : Boolean`

Visibility: default

Description: Frees a previously allocated memory block.

Parameters:

blockhandle the handle to the memory area to free.

Return value: `True` if successful, `false` otherwise. Notes: Frees memory that was previously allocated with `allocate_memory_block` (465) . This function doesn't free any descriptors mapped to this block, it's the application's responsibility.

Errors: Check `int31error` (463) variable.

See also: `allocate_memory_block` (465)

free_rm_callback

Synopsis: Release real mode callback.

Declaration: `function free_rm_callback(var intaddr: tseginfo) : Boolean`

Visibility: default

Description: Releases a real mode callback address that was previously allocated with the `get_rm_callback` (473) function.

Parameters:

intaddr real mode address buffer returned by `get_rm_callback` (473) .

Return values: `True` if successful, `False` if not

For an example, see `get_rm_callback` (473).

Errors: Check the `int31error` (463) variable.

See also: `set_rm_interrupt` (488), `get_rm_callback` (473)

get_cs

Synopsis: Get CS selector

Declaration: `function get_cs : Word`

Visibility: default

Description: Returns the cs selector.

Return value: The content of the cs segment register.

For an example, see `set_pm_interrupt` (487).

Errors: None.

See also: `get_ds` (470), `get_ss` (478)

get_descriptor_access_right

Synopsis: Get descriptor's access rights

Declaration: `function get_descriptor_access_right(d: Word) : LongInt`

Visibility: default

Description: Gets the access rights of a descriptor.

Parameters:

d selector to descriptor.

Return value: Access rights bit field.

Errors: Check the `int31error` ([463](#)) variable.

See also: `set_descriptor_access_right` ([487](#))

get_ds

Synopsis: Get DS Selector

Declaration: `function get_ds : Word`

Visibility: default

Description: Returns the ds selector.

Return values: The content of the ds segment register.

Errors: None.

See also: `get_cs` ([469](#)), `get_ss` ([478](#))

get_exception_handler

Declaration: `function get_exception_handler(e: Byte; var intaddr: tseginfo) : Boolean`

Visibility: default

get_linear_addr

Synopsis: Convert physical to linear address

Declaration: `function get_linear_addr(phys_addr: LongInt; size: LongInt) : LongInt`

Visibility: default

Description: Converts a physical address into a linear address.

Parameters:

phys_addr physical address of device.

size Size of region to map in bytes.

Return value: Linear address that can be used to access the physical memory. Notes: It's the applications responsibility to allocate and set up a descriptor for access to the memory. This function shouldn't be used to map real mode addresses.

Errors: Check the `int31error` ([463](#)) variable.

See also: `allocate_ldt_descriptors` ([463](#)), `set_segment_limit` ([489](#)), `set_segment_base_address` ([489](#))

get_meminfo

Synopsis: Return information on the available memory

Declaration: `function get_meminfo(var meminfo: tmeminfo) : Boolean`

Visibility: default

Description: Returns information about the amount of available physical memory, linear address space, and disk space for page swapping.

Parameters:

meminfobuffer to fill memory information into.

Return values: Due to an implementation bug this function always returns `False`, but it always succeeds.

Remark: Notes: Only the first field of the returned structure is guaranteed to contain a valid value. Any fields that are not supported by the DPMI host will be set by the host to `-1` (`0FFFFFFFFH`) to indicate that the information is not available. The size of the pages used by the DPMI host can be obtained with the `get_page_size` (472) function.

Errors: Check the `int31error` (463) variable.

See also: `get_page_size` (472)

Listing: `./go32ex/meminfo.pp`

```

uses
    go32;

var
    meminfo : tmeminfo;

begin
    get_meminfo(meminfo);
    if (int31error <> 0) then begin
        Writeln('Error getting DPMI memory information... Halting');
        Writeln('DPMI error number : ', int31error);
    end else begin
        with meminfo do begin
            Writeln('Largest available free block : ',
                available_memory div 1024, ' kbytes');
            if (available_pages <> -1) then
                Writeln('Maximum available unlocked pages : ',
                    available_pages);
            if (available_lockable_pages <> -1) then
                Writeln('Maximum lockable available pages : ',
                    available_lockable_pages);
            if (linear_space <> -1) then
                Writeln('Linear address space size : ',
                    linear_space*get_page_size div 1024, ' kbytes');
            if (unlocked_pages <> -1) then
                Writeln('Total number of unlocked pages : ',
                    unlocked_pages);
            if (available_physical_pages <> -1) then
                Writeln('Total number of free pages : ',
                    available_physical_pages);
            if (total_physical_pages <> -1) then
                Writeln('Total number of physical pages : ',

```

```

                                total_physical_pages);
    if (free_linear_space <> -1) then
        WriteLn('Free linear address space : ',
                free_linear_space*get_page_size div 1024,
                ' kbytes');
    if (max_pages_in_paging_file <> -1) then
        WriteLn('Maximum size of paging file : ',
                max_pages_in_paging_file*get_page_size div 1024,
                ' kbytes');
    end;
end;
end.
```

get_next_selector_increment_value

Synopsis: Return selector increment value

Declaration: function get_next_selector_increment_value : Word

Visibility: default

Description: Returns the selector increment value when allocating multiple subsequent descriptors via allocate_ldt_descriptors (463).

Return value: Selector increment value.

Remark: Notes: Because allocate_ldt_descriptors (463) only returns the selector for the first descriptor and so the value returned by this function can be used to calculate the selectors for subsequent descriptors in the array.

Errors: Check the int31error (463) variable.

See also: allocate_ldt_descriptors (463), free_ldt_descriptor (468)

get_page_size

Synopsis: Return the page size

Declaration: function get_page_size : LongInt

Visibility: default

Description: Returns the size of a single memory page.

Return value: Size of a single page in bytes.

Remark: The returned size is typically 4096 bytes.

For an example, see get_meminfo (471).

Errors: Check the int31error (463) variable.

See also: get_meminfo (471)

get_pm_exception_handler

Declaration: function get_pm_exception_handler(e: Byte;var intaddr: tseginfo)
: Boolean

Visibility: default

get_pm_interrupt

Synopsis: Return protected mode interrupt handler

Declaration: `function get_pm_interrupt(vector: Byte; var intaddr: tseginfo) : Boolean`

Visibility: default

Description: Returns the address of a current protected mode interrupt handler.

Parameters:

vector interrupt handler number you want the address to.

intaddr buffer to store address.

Return values: True if successful, False if not.

Remark: The returned address is a protected mode selector:offset address.

For an example, see `set_pm_interrupt (487)`.

Errors: Check the `int31error (463)` variable.

See also: `set_pm_interrupt (487)`, `set_rm_interrupt (488)`, `get_rm_interrupt (476)`

get_rm_callback

Synopsis: Return real mode callback

Declaration: `function get_rm_callback(pm_func: pointer; const reg: trealregs;
var rmcb: tseginfo) : Boolean`

Visibility: default

Description: Returns a unique real mode `segment:offset` address, known as a "real mode callback," that will transfer control from real mode to a protected mode procedure.

Parameters:

pm_func pointer to the protected mode callback function.

regs supplied registers structure.

rmcb buffer to real mode address of callback function.

Return values: True if successful, otherwise False.

Remark: Callback addresses obtained with this function can be passed by a protected mode program for example to an interrupt handler, device driver, or TSR, so that the real mode program can call procedures within the protected mode program or notify the protected mode program of an event. The contents of the supplied `regs` structure is not valid after function call, but only at the time of the actual callback.

Errors: Check the `int31error (463)` variable.

See also: `free_rm_callback (469)`

Listing: `./go32ex/callback.pp`

```
{$ASMMODE ATT}
{$MODE FPC}
```

```
uses
```

```
    crt ,
    go32;
```

```
const
```

```
    mouseint = $33;
```

```
var
```

```
    mouse_regs      : trealregs; external name '___v2prt0_rmcb_regs';
    mouse_seginfo    : tseginfo;
```

```
var
```

```
    mouse_numbuttons : longint;

    mouse_action      : word;
    mouse_x, mouse_y  : Word;
    mouse_b           : Word;

    userproc_installed : Longbool;
    userproc_length    : Longint;
    userproc_proc      : pointer;
```

```
procedure callback_handler; assembler;
```

```
asm
```

```
    pushw %ds
    pushl %eax
    movw %es, %ax
    movw %ax, %ds
```

```
    cmpl $1, USERPROC_INSTALLED
    jne .LNoCallback
    pushal
    movw DOSmemSELECTOR, %ax
    movw %ax, %fs
    call *USERPROC_PROC
    popal
```

```
.LNoCallback:
```

```
    popl %eax
    popw %ds
```

```
    pushl %eax
    movl (%esi), %eax
    movl %eax, %es: 42(%edi)
    addw $4, %es:46(%edi)
    popl %eax
    iret
```

```
end;
```

```
procedure mouse_dummy; begin end;
```

```
procedure textuserproc;
```

```
begin
```

```
    mouse_b := mouse_regs.bx;
    mouse_x := (mouse_regs.cx shr 3) + 1;
    mouse_y := (mouse_regs.dx shr 3) + 1;
```

```

end;

procedure install_mouse(userproc : pointer; userproclen : longint);
var r : trealregs;
begin
    r.eax := $0; realintr(mouseint, r);
    if (r.eax <> $FFFF) then begin
        Writeln('No Microsoft compatible mouse found');
        Writeln('A Microsoft compatible mouse driver is necessary ',
            'to run this example');
        halt;
    end;
    if (r.bx = $ffff) then mouse_numbuttons := 2
    else mouse_numbuttons := r.bx;
    Writeln(mouse_numbuttons, ' button Microsoft compatible mouse ',
        ' found. ');
    if (userproc <> nil) then begin
        userproc_proc := userproc;
        userproc_installed := true;
        userproc_length := userproclen;
        lock_code(userproc_proc, userproc_length);
    end else begin
        userproc_proc := nil;
        userproc_length := 0;
        userproc_installed := false;
    end;
    lock_data(mouse_x, sizeof(mouse_x));
    lock_data(mouse_y, sizeof(mouse_y));
    lock_data(mouse_b, sizeof(mouse_b));
    lock_data(mouse_action, sizeof(mouse_action));

    lock_data(userproc_installed, sizeof(userproc_installed));
    lock_data(userproc_proc, sizeof(userproc_proc));

    lock_data(mouse_regs, sizeof(mouse_regs));
    lock_data(mouse_seginfo, sizeof(mouse_seginfo));
    lock_code(@callback_handler,
        longint(@mouse_dummy) - longint(@callback_handler));
    get_rm_callback(@callback_handler, mouse_regs, mouse_seginfo);
    r.eax := $0c; r.ecx := $7f;
    r.edx := longint(mouse_seginfo.offset);
    r.es := mouse_seginfo.segment;
    realintr(mouseint, r);
    r.eax := $01;
    realintr(mouseint, r);
end;

procedure remove_mouse;
var
    r : trealregs;
begin
    r.eax := $02; realintr(mouseint, r);
    r.eax := $0c; r.ecx := 0; r.edx := 0; r.es := 0;
    realintr(mouseint, r);
    free_rm_callback(mouse_seginfo);
    if (userproc_installed) then begin
        unlock_code(userproc_proc, userproc_length);
        userproc_proc := nil;
    end;

```

```

        userproc_length := 0;
        userproc_installed := false;
    end;
    unlock_data(mouse_x, sizeof(mouse_x));
    unlock_data(mouse_y, sizeof(mouse_y));
    unlock_data(mouse_b, sizeof(mouse_b));
    unlock_data(mouse_action, sizeof(mouse_action));

    unlock_data(userproc_proc, sizeof(userproc_proc));
    unlock_data(userproc_installed, sizeof(userproc_installed));

    unlock_data(mouse_regs, sizeof(mouse_regs));
    unlock_data(mouse_seginf, sizeof(mouse_seginf));
    unlock_code(@callback_handler,
        longint(@mouse_dummy) - longint(@callback_handler));
    fillchar(mouse_seginf, sizeof(mouse_seginf), 0);
end;

begin
    install_mouse(@textuserproc, 400);
    Writeln('Press any key to exit...');
    while (not keypressed) do begin
        gotoxy(1, wherey);
        write('MouseX : ', mouse_x:2, ' MouseY : ', mouse_y:2,
            ' Buttons : ', mouse_b:2);
    end;
    remove_mouse;
end.

```

get_rm_interrupt

Synopsis: Get real mode interrupt vector

Declaration: function get_rm_interrupt(vector: Byte; var intaddr: tseginf) : Boolean

Visibility: default

Description: Returns the contents of the current machine's real mode interrupt vector for the specified interrupt.

Parameters:

vector interrupt vector number.

intaddr buffer to store real mode segment:offset address.

Return values: True if successful, False otherwise.

Remark: The returned address is a real mode segment address, which isn't valid in protected mode.

Errors: Check the int31error ([463](#)) variable.

See also: set_rm_interrupt ([488](#)), set_pm_interrupt ([487](#)), get_pm_interrupt ([473](#))

get_run_mode

Synopsis: Return current run mode

Declaration: function get_run_mode : Word

Visibility: default

Description: Returns the current mode your application runs with.

Return values: One of the constants used by this function.

Errors: None.

See also: `get_run_mode` ([476](#))

Listing: `./go32ex/getrunmd.pp`

```

uses
    go32;

begin
    case (get_run_mode) of
        rm_unknown :
            WriteLn( 'Unknown environment found' );
        rm_raw      :
            WriteLn( 'You are currently running in raw mode ',
                    '(without HIMEM)' );
        rm_xms      :
            WriteLn( 'You are currently using HIMEM.SYS only' );
        rm_vcpi     :
            WriteLn( 'VCPI server detected. You''re using HIMEM and ',
                    'EMM386' );
        rm_dpml     :
            WriteLn( 'DPML detected. You''re using a DPML host like ',
                    'a windows DOS box or CWSDPML' );
    end;
end.
```

get_segment_base_address

Synopsis: Return base address from descriptor table

Declaration: `function get_segment_base_address(d: Word) : LongInt`

Visibility: default

Description: Returns the 32-bit linear base address from the descriptor table for the specified segment.

Parameters:

`d` selector of the descriptor you want the base address of.

Return values: Linear base address of specified descriptor.

For an example, see `allocate_ldt_descriptors` ([463](#)).

Errors: Check the `int31error` ([463](#)) variable.

See also: `allocate_ldt_descriptors` ([463](#)), `set_segment_base_address` ([489](#)), `allocate_ldt_descriptors` ([463](#)), `set_segment_limit` ([489](#)), `get_segment_limit` ([478](#))

get_segment_limit

Synopsis: Return segment limite from descriptor

Declaration: `function get_segment_limit(d: Word) : LongInt`

Visibility: default

Description: Returns a descriptors segment limit.

Parameters:

dselector.

Return value: Limit of the descriptor in bytes.

Errors: Returns zero if descriptor is invalid.

See also: `allocate_ldt_descriptors` (463), `set_segment_limit` (489), `set_segment_base_address` (489), `get_segment_base_address` (477)

get_ss

Synopsis: Return SS selector

Declaration: `function get_ss : Word`

Visibility: default

Description: Returns the ss selector.

Return values: The content of the ss segment register.

Errors: None.

See also: `get_ds` (470), `get_cs` (469)

global_dos_alloc

Synopsis: Allocate DOS real mode memory

Declaration: `function global_dos_alloc(bytes: LongInt) : LongInt`

Visibility: default

Description: Allocates a block of dos real mode memory.

Parameters:

bytessize of requested real mode memory.

Return values: The low word of the returned value contains the selector to the allocated dos memory block, the high word the corresponding real mode segment value. The offset value is always zero. This function allocates memory from dos memory pool, i.e. memory below the 1 MB boundary that is controlled by dos. Such memory blocks are typically used to exchange data with real mode programs, TSRs, or device drivers. The function returns both the real mode segment base address of the block and one descriptor that can be used by protected mode applications to access the block. This function should only used for temporary buffers to get real mode information (e.g. interrupts that need a data structure in ES:(E)DI), because every single block needs an unique selector. The returned selector should only be freed by a `global_dos_free` (480) call.

Errors: Check the `int31error` (463) variable.

See also: `global_dos_free` (480)

Listing: `./go32ex/buffer.pp`

```

uses
    go32;

procedure dosalloc(var selector : word;
    var segment : word; size : longint);
var
    res : longint;
begin
    res := global_dos_alloc(size);
    selector := word(res);
    segment := word(res shr 16);
end;

procedure dosfree(selector : word);
begin
    global_dos_free(selector);
end;

type
    VBEInfoBuf = packed record
        Signature : array[0..3] of char;
        Version : Word;
        reserved : array[0..505] of byte;
    end;

var
    selector ,
    segment : Word;

    r : trealregs;
    infobuf : VBEInfoBuf;

begin
    fillchar(r, sizeof(r), 0);
    fillchar(infobuf, sizeof(VBEInfoBuf), 0);
    dosalloc(selector, segment, sizeof(VBEInfoBuf));
    if (int31error <> 0) then begin
        Writeln('Error while allocating real mode memory, halting');
        halt;
    end;
    infobuf.Signature := 'VBE2';
    dosmemput(segment, 0, infobuf, sizeof(infobuf));
    r.ax := $4f00; r.es := segment;
    realintr($10, r);
    dosmemget(segment, 0, infobuf, sizeof(infobuf));
    dosfree(selector);
    if (r.ax <> $4f) then begin
        Writeln('VBE BIOS extension not available, function call ',
            'failed');
        halt;
    end;
    if (infobuf.signature[0] = 'V') and
        (infobuf.signature[1] = 'E') and

```

```

        (infobuf.signature[2] = 'S') and
        (infobuf.signature[3] = 'A') then begin
            WriteLn('VBE version ', hi(infobuf.version), '.',',
                    lo(infobuf.version), ' detected');
        end;
    end.

```

global_dos_free

Synopsis: Free DOS memory block

Declaration: function global_dos_free(selector: Word) : Boolean

Visibility: default

Description: Frees a previously allocated dos memory block.

Parameters:

selector selector to the dos memory block.

Return value: True if successful, False otherwise.

Remark: The descriptor allocated for the memory block is automatically freed and hence invalid for further use. This function should only be used for memory allocated by global_dos_alloc (478).

For an example, see global_dos_alloc (478).

Errors: Check the int31error (463) variable.

See also: global_dos_alloc (478)

inportb

Synopsis: Read byte from I/O port

Declaration: function inportb(port: Word) : Byte

Visibility: default

Description: Reads 1 byte from the selected I/O port.

Parameters:

port the I/O port number which is read.

Return values: Current I/O port value.

Errors: None.

See also: outportb (483), inportw (481), inportl (481)

inportl

Synopsis: Read longint from I/O port

Declaration: `function inportl(port: Word) : LongInt`

Visibility: default

Description: Reads 1 longint from the selected I/O port.

Parameters:

port the I/O port number which is read.

Return values: Current I/O port value.

Errors: None.

See also: [outportb \(483\)](#), [inportb \(480\)](#), [inportw \(481\)](#)

inportw

Synopsis: Read word from I/O port

Declaration: `function inportw(port: Word) : Word`

Visibility: default

Description: Reads 1 word from the selected I/O port.

Parameters:

port the I/O port number which is read.

Return values: Current I/O port value.

Errors: None.

See also: [outportw \(484\)](#), [inportb \(480\)](#), [inportl \(481\)](#)

lock_code

Synopsis: Lock code memory range

Declaration: `function lock_code(functionaddr: pointer; size: LongInt) : Boolean`

Visibility: default

Description: Locks a memory range which is in the code segment selector.

Parameters:

functionaddr address of the function to be locked.

size size in bytes to be locked.

Return values: True if successful, False otherwise.

For an example, see [get_rm_callback \(473\)](#).

Errors: Check the [int31error \(463\)](#) variable.

See also: [lock_linear_region \(482\)](#), [lock_data \(482\)](#), [unlock_linear_region \(491\)](#), [unlock_data \(491\)](#), [unlock_code \(490\)](#)

lock_data

Synopsis: Lock data memory range

Declaration: `function lock_data(var data;size: LongInt) : Boolean`

Visibility: default

Description: Locks a memory range which resides in the data segment selector.

Parameters:

data address of data to be locked.

size length of data to be locked.

Return values: True if successful, False otherwise.

For an example, see `get_rm_callback` (473).

Errors: Check the `int31error` (463) variable.

See also: `lock_linear_region` (482), `lock_code` (481), `unlock_linear_region` (491), `unlock_data` (491), `unlock_code` (490)

lock_linear_region

Synopsis: Lock linear memory region

Declaration: `function lock_linear_region(linearaddr: LongInt;size: LongInt) : Boolean`

Visibility: default

Description: Locks a memory region to prevent swapping of it.

Parameters:

linearaddr the linear address of the memory are to be locked.

size size in bytes to be locked.

Return value: True if successful, False otherwise.

Errors: Check the `int31error` (463) variable.

See also: `lock_data` (482), `lock_code` (481), `unlock_linear_region` (491), `unlock_data` (491), `unlock_code` (490)

map_device_in_memory_block

Declaration: `function map_device_in_memory_block(handle: LongInt;offset: LongInt;
pagecount: LongInt;device: LongInt)
: Boolean`

Visibility: default

outportb

Synopsis: Write byte to I/O port

Declaration: `procedure outportb(port: Word; data: Byte)`

Visibility: default

Description: Sends 1 byte of data to the specified I/O port.

Parameters:

port the I/O port number to send data to.

data value sent to I/O port.

Return values: None.

Errors: None.

See also: `inportb` ([480](#)), `outportl` ([483](#)), `outportw` ([484](#))

Listing: `./go32ex/outport.pp`

uses

`crt ,
go32;`

begin

`outportb($61, $ff);
delay(50);
outportb($61, $0);`

end.

outportl

Synopsis: Write longint to I/O port

Declaration: `procedure outportl(port: Word; data: LongInt)`

Visibility: default

Description: Sends 1 longint of data to the specified I/O port.

Parameters:

port the I/O port number to send data to.

data value sent to I/O port.

Return values: None.

For an example, see `outportb` ([483](#)).

Errors: None.

See also: `inportl` ([481](#)), `outportw` ([484](#)), `outportb` ([483](#))

outportw

Synopsis: Write word to I/O port

Declaration: `procedure outportw(port: Word; data: Word)`

Visibility: default

Description: Sends 1 word of data to the specified I/O port.

Parameters:

port the I/O port number to send data to.

data value sent to I/O port.

Return values: None.

For an example, see `outportb` (483).

Errors: None.

See also: `inportw` (481), `outportl` (483), `outportb` (483)

realintr

Synopsis: Simulate interrupt

Declaration: `function realintr(intnr: Word; var regs: trealregs) : Boolean`

Visibility: default

Description: Simulates an interrupt in real mode.

Parameters:

intnr interrupt number to issue in real mode.

regs registers data structure.

Return values: The supplied registers data structure contains the values that were returned by the real mode interrupt. `True` if successful, `False` if not.

Remark: The function transfers control to the address specified by the real mode interrupt vector of `intnr`. The real mode handler must return by executing an `IRET`.

Errors: Check the `int31error` (463) variable.

Listing: `./go32ex/flags.pp`

uses

`go32;`

var

`r : trealregs;`

begin

`r.ax := $5300;`

`r.bx := 0;`

`realintr($15, r);`

if `((r.flags and carryflag)=0)` **then begin**

`WriteLn('APM v', (r.ah and $f), '.',`

`(r.al shr 4), (r.al and $f), ' detected');`

end else

`WriteLn('APM not present');`

end.

request_linear_region

Declaration: `function request_linear_region(linearaddr: LongInt; size: LongInt;
var blockhandle: LongInt) : Boolean`

Visibility: default

segment_to_descriptor

Synopsis: Map segment address to descriptor

Declaration: `function segment_to_descriptor(seg: Word) : Word`

Visibility: default

Description: Maps a real mode segment (paragraph) address onto an descriptor that can be used by a protected mode program to access the same memory.

Parameters:

seg the real mode segment you want the descriptor to.

Return values: Descriptor to real mode segment address.

Remark: The returned descriptors limit will be set to 64 kB. Multiple calls to this function with the same segment address will return the same selector. Descriptors created by this function can never be modified or freed. Programs which need to examine various real mode addresses using the same selector should use the function `allocate_ldt_descriptors` (463) and change the base address as necessary.

For an example, see `seg_fillchar` (485).

Errors: Check the `int31error` (463) variable.

See also: `allocate_ldt_descriptors` (463), `free_ldt_descriptor` (468), `set_segment_base_address` (489)

seg_fillchar

Synopsis: Fill segment with byte value

Declaration: `procedure seg_fillchar(seg: Word; ofs: LongInt; count: LongInt; c: Char)`

Visibility: default

Description: Sets a memory area to a specific value.

Parameters:

seg selector to memory area.

ofs offset to memory.

count number of bytes to set.

c byte data which is set.

Return values: None.

Notes: No range check is done in any way.

Errors: None.

See also: `seg_move` (486), `seg_fillword` (486), `dosmemfillchar` (460), `dosmemfillword` (460), `dosmemget` (461), `dosmemput` (461), `dosmemmove` (461)

Listing: ./go32ex/vgasel.pp

```

uses
    go32;

var
    vgasel : Word;
    r : treatregs;

begin
    r.eax := $13; realintr($10, r);
    vgasel := segment_to_descriptor($A000);
    seg_fillchar(vgasel, 0, 64000, #15);
    readln;
    r.eax := $3; realintr($10, r);
end .

```

seg_fillword

Synopsis: Fill segment with word value

Declaration: `procedure seg_fillword(seg: Word; ofs: LongInt; count: LongInt; w: Word)`

Visibility: default

Description: Sets a memory area to a specific value.

Parameters:

seg selector to memory area.

ofs offset to memory.

count number of words to set.

w word data which is set.

Return values: None.

Notes: No range check is done in any way.

For an example, see `allocate_ldt_descriptors` ([463](#)).

Errors: None.

See also: `seg_move` ([486](#)), `seg_fillchar` ([485](#)), `dosmemfillchar` ([460](#)), `dosmemfillword` ([460](#)), `dosmemget` ([461](#)), `dosmemput` ([461](#)), `dosmemmove` ([461](#))

seg_move

Synopsis: Move data between 2 locations

Declaration: `procedure seg_move(sseg: Word; source: LongInt; dseg: Word; dest: LongInt; count: LongInt)`

Visibility: default

Description: Copies data between two memory locations.

Parameters:

sseg source selector.

sourcesource offset.

dsegdestination selector.

destdestination offset.

countsize in bytes to copy.

Return values: None.

Remark: Overlapping is only checked if the source selector is equal to the destination selector. No range check is done.

For an example, see `allocate_ldt_descriptors` (463).

Errors: None.

See also: `seg_fillchar` (485), `seg_fillword` (486), `dosmemfillchar` (460), `dosmemfillword` (460), `dosmemget` (461), `dosmemput` (461), `dosmemmove` (461)

set_descriptor_access_right

Declaration: `function set_descriptor_access_right(d: Word;w: Word) : LongInt`

Visibility: default

set_exception_handler

Declaration: `function set_exception_handler(e: Byte;const intaddr: tseginfo) : Boolean`

Visibility: default

set_pm_exception_handler

Declaration: `function set_pm_exception_handler(e: Byte;const intaddr: tseginfo) : Boolean`

Visibility: default

set_pm_interrupt

Synopsis: Set protected mode interrupt handler

Declaration: `function set_pm_interrupt(vector: Byte;const intaddr: tseginfo) : Boolean`

Visibility: default

Description: Sets the address of the protected mode handler for an interrupt.

Parameters:

vectornumber of protected mode interrupt to set.

intaddrselector:offset address to the interrupt vector.

Return values: True if successful, False otherwise.

Remark: The address supplied must be a valid `selector:offset` protected mode address.

Errors: Check the int31error (463) variable.

See also: [get_pm_interrupt \(473\)](#), [set_rm_interrupt \(488\)](#), [get_rm_interrupt \(476\)](#)

Listing: ./go32ex/intpm.pp

```

uses
    crt ,
    go32;

const
    int1c = $1c;

var
    oldint1c : tseginfo;
    newint1c : tseginfo;

    int1c_counter : Longint;

    int1c_ds : Word; external name '___v2prt0_ds_alias';

procedure int1c_handler; assembler;
asm
    cli
    pushw %ds
    pushw %ax
    movw %cs:int1c_ds, %ax
    movw %ax, %ds
    incl int1c_counter
    popw %ax
    popw %ds
    sti
    iret
end;

var i : Longint;

begin
    newint1c.offset := @int1c_handler;
    newint1c.segment := get_cs;
    get_pm_interrupt(int1c, oldint1c);
    WriteLn('-- Press any key to exit --');
    set_pm_interrupt(int1c, newint1c);
    while (not keypressed) do begin
        gotoxy(1, wherey);
        write('Number of interrupts occurred : ', int1c_counter);
    end;
    set_pm_interrupt(int1c, oldint1c);
end.

```

set_rm_interrupt

Synopsis: Set real mode interrupt handler

```
Declaration: function set_rm_interrupt(vector: Byte; const intaddr: tseginfo)
                                         : Boolean
```

Visibility: default

Description: Sets a real mode interrupt handler.

Parameters:

vector the interrupt vector number to set.

intaddr address of new interrupt vector.

Return values: True if successful, otherwise False.

Remark: The address supplied MUST be a real mode segment address, not a `selector:offset` address. So the interrupt handler must either reside in dos memory (below 1 Mb boundary) or the application must allocate a real mode callback address with `get_rm_callback` (473).

Errors: Check the `int31error` (463) variable.

See also: `get_rm_interrupt` (476), `set_pm_interrupt` (487), `get_pm_interrupt` (473), `get_rm_callback` (473)

set_segment_base_address

Synopsis: Set descriptor's base address

Declaration: `function set_segment_base_address(d: Word; s: LongInt) : Boolean`

Visibility: default

Description: Sets the 32-bit linear base address of a descriptor.

Parameters:

d selector.

s new base address of the descriptor.

Errors: Check the `int31error` (463) variable.

See also: `allocate_ldt_descriptors` (463), `get_segment_base_address` (477), `allocate_ldt_descriptors` (463), `set_segment_limit` (489), `get_segment_base_address` (477), `get_segment_limit` (478)

set_segment_limit

Synopsis: Set descriptor limit

Declaration: `function set_segment_limit(d: Word; s: LongInt) : Boolean`

Visibility: default

Description: Sets the limit of a descriptor.

Parameters:

d selector.

s new limit of the descriptor.

Return values: Returns True if successful, else False.

Remark: The new limit specified must be the byte length of the segment - 1. Segment limits bigger than or equal to 1MB must be page aligned, they must have the lower 12 bits set.

For an example, see `allocate_ldt_descriptors` (463).

Errors: Check the `int31error` (463) variable.

See also: `allocate_ldt_descriptors` (463), `set_segment_base_address` (489), `get_segment_limit` (478), `set_segment_limit` (489)

tb_offset

Declaration: `function tb_offset : LongInt`

Visibility: default

tb_segment

Declaration: `function tb_segment : LongInt`

Visibility: default

tb_size

Synopsis: Return DOS transfer memory buffer size

Declaration: `function tb_size : LongInt`

Visibility: default

Description: Returns the size of the pre-allocated dos memory buffer.

Return values: The size of the pre-allocated dos memory buffer. This block always seems to be 16k in size, but don't rely on this.

Errors: None.

See also: `transfer_buffer` ([490](#)), `copyfromdos` ([466](#)), `copytodos` ([466](#))

transfer_buffer

Synopsis: Return offset of DOS transfer buffer

Declaration: `function transfer_buffer : LongInt`

Visibility: default

Description: `transfer_buffer` returns the offset of the transfer buffer.

Errors: None.

See also: `tb_size` ([490](#))

unlock_code

Synopsis: Unlock code segment

Declaration: `function unlock_code(functionaddr: pointer;size: LongInt) : Boolean`

Visibility: default

Description: Unlocks a memory range which resides in the code segment selector.

Parameters:

functionaddr address of function to be unlocked.

size size bytes to be unlocked.

Return value: `True` if successful, `False` otherwise.

For an example, see `get_rm_callback` (473).

Errors: Check the `int31error` (463) variable.

See also: `unlock_linear_region` (491), `unlock_data` (491), `lock_linear_region` (482), `lock_data` (482), `lock_code` (481)

unlock_data

Synopsis: Unlock data segment

Declaration: `function unlock_data(var data;size: LongInt) : Boolean`

Visibility: default

Description: Unlocks a memory range which resides in the data segment selector.

Parameters:

data address of memory to be unlocked.

size size bytes to be unlocked.

Return values: `True` if successful, `False` otherwise.

For an example, see `get_rm_callback` (473).

Errors: Check the `int31error` (463) variable.

See also: `unlock_linear_region` (491), `unlock_code` (490), `lock_linear_region` (482), `lock_data` (482), `lock_code` (481)

unlock_linear_region

Synopsis: Unlock linear memory region

Declaration: `function unlock_linear_region(linearaddr: LongInt;size: LongInt) : Boolean`

Visibility: default

Description: Unlocks a previously locked linear region range to allow it to be swapped out again if needed.

Parameters:

linearaddr linear address of the memory to be unlocked.

size size bytes to be unlocked.

Return values: `True` if successful, `False` otherwise.

Errors: Check the `int31error` (463) variable.

See also: `unlock_data` (491), `unlock_code` (490), `lock_linear_region` (482), `lock_data` (482), `lock_code` (481)

Chapter 11

Reference for unit 'gpm'

11.1 Used units

Table 11.1: Used units by unit 'gpm'

Name	Page
baseUnix	492

11.2 Overview

The GPM unit implements an interface to `libgpm`, the console program for mouse handling. This unit was created by Peter Vreman, and is only available on linux.

When this unit is used, your program is linked to the C libraries, so you must take care of the C library version. Also, it will only work with version 1.17 or higher of the `libgpm` library.

11.3 Constants, types and variables

Constants

`GPM_BOT` = 2

Bottom of area.

`GPM_B_LEFT` = 4

Left mouse button identifier.

`GPM_B_MIDDLE` = 2

Middle mouse button identifier.

`GPM_B_RIGHT` = 1

Right mouse button identifier.

GPM_DOUBLE = 32

Mouse double click event.

GPM_DOWN = 4

Mouse button down event.

GPM_DRAG = 2

Mouse drag event.

GPM_ENTER = 512

Enter area event.

GPM_HARD = 256

?

GPM_LEAVE = 1024

Leave area event.

GPM_LEFT = 4

Left side of area.

GPM_MAGIC = \$47706D4C

GPM_MFLAG = 128

Motion flag.

GPM_MOVE = 1

Mouse move event.

GPM_NODE_CTL = GPM_NODE_DEV

Control socket

GPM_NODE_DEV = '/dev/gpmctl'

Device socket filename

GPM_NODE_DIR = _PATH_VARRUN

Where to write socket.

GPM_NODE_DIR_MODE = 0775

Mode of socket.

GPM_NODE_FIFO = '/dev/gpmdata'

FIFO name

GPM_NODE_PID = '/var/run/gpm.pid'

Name of PID file.

GPM_RGT = 8

Right side of area.

GPM_SINGLE = 16

Mouse single click event.

GPM_TOP = 1

Top of area.

GPM_TRIPLE = 64

Mouse triple click event.

GPM_UP = 8

Mouse button up event.

_PATH_DEV = '/dev/'

Location of /dev directory.

_PATH_VARRUN = '/var/run/'

Location of run PID files directory.

Types

Pgpmconnect = Pgpm_connect

Pointer to TGpmConnect ([495](#)) record.

Pgpmevent = Pgpm_event

Pointer to TGpmEvent ([496](#)) record

Pgpmroi = Pgpm_roi

Pointer to TGpmRoi ([496](#)) record.

Pgpm_connect = ^TGpm_connect

Pointer to TGpm_Connect (495) record.

```
Pgpm_event = ^Tgpm_event
```

Pointer to TGpm_Event (496) record

```
Pgpm_roi = ^Tgpm_roi
```

Pointer to Tgpm_roi (496) record.

```
Tgpmconnect = Tgpm_connect
```

Alias for TGpm_Connect (495) record.

```
TGpmEtype = LongInt
```

Type for event type.

```
Tgpmevent = Tgpm_event
```

Alias for TGPM_EVent (496) record

```
TGpmHandler = function(var event: Tgpmevent; clientdata: pointer)
                  : LongInt
```

Mouse event handler callback.

```
TGpmMargin = LongInt
```

Type to hold area margin.

```
Tgpmroi = Tgpm_roi
```

Alias for TGpm_roi (496)Record

```
Tgpm_connect = record
  eventMask : Word;
  defaultMask : Word;
  minMod : Word;
  maxMod : Word;
  pid : LongInt;
  vc : LongInt;
end
```

GPM server connection information.

```
Tgpm_event = record
  buttons : Byte;
  modifiers : Byte;
  vc : Word;
  dx : Word;
  dy : Word;
  x : Word;
```

```

y : Word;
wdx : Word;
wdy : Word;
EventType : TGpmEtype;
clicks : LongInt;
margin : TGpmMargin;
end

```

Tgpm_event describes the events that are reported by GPM.

```

Tgpm_roi = record
  xmin : Integer;
  xmax : Integer;
  ymin : Integer;
  ymax : Integer;
  minmod : Word;
  maxmod : Word;
  eventmask : Word;
  owned : Word;
  handler : TGpmHandler;
  clientdata : pointer;
  prev : Pgpm_roi;
  next : Pgpm_roi;
end

```

Record used to define regions of interest.

Variables

gpm_current_roi : Pgpm_roi

Internal gpm library variable. Do not use.

gpm_handler : TGpmHandler

Internal gpm library variable. Do not use.

gpm_roi : Pgpm_roi

Internal gpm library variable. Do not use.

gpm_roi_data : pointer

Internal gpm library variable. Do not use.

gpm_roi_handler : TGpmHandler

Internal gpm library variable. Do not use.

11.4 Procedures and functions

Gpm_AnyDouble

Synopsis: Check whether event has double click event.

Declaration: `function Gpm_AnyDouble(EventType: LongInt) : Boolean`

Visibility: default

Description: `Gpm_AnyDouble` returns `True` if `EventType` contains the `GPM_DOUBLE` flag, `False` otherwise.

Errors: None.

See also: `Gpm_StrictSingle` (502), `Gpm_AnySingle` (497), `Gpm_StrictDouble` (501), `Gpm_StrictTriple` (502), `Gpm_AnyTriple` (497)

Gpm_AnySingle

Synopsis: Check whether event has a single click event.

Declaration: `function Gpm_AnySingle(EventType: LongInt) : Boolean`

Visibility: default

Description: `Gpm_AnySingle` returns `True` if `EventType` contains the `GPM_SINGLE` flag, `False` otherwise.

Errors: None.

See also: `Gpm_StrictSingle` (502), `Gpm_AnyDouble` (497), `Gpm_StrictDouble` (501), `Gpm_StrictTriple` (502), `Gpm_AnyTriple` (497)

Gpm_AnyTriple

Synopsis: Check whether event has a triple click event.

Declaration: `function Gpm_AnyTriple(EventType: LongInt) : Boolean`

Visibility: default

Description: `Gpm_AnySingle` returns `True` if `EventType` contains the `GPM_TRIPLE` flag, `False` otherwise.

Errors: None.

See also: `Gpm_StrictSingle` (502), `Gpm_AnyDouble` (497), `Gpm_StrictDouble` (501), `Gpm_StrictTriple` (502), `Gpm_AnySingle` (497)

gpm_close

Synopsis: Close connection to GPM server.

Declaration: `function gpm_close : LongInt`

Visibility: default

Description: `Gpm_Close` closes the current connection, and pops the connection stack; this means that the previous connection becomes active again.

The function returns -1 if the current connection is not the last one, and it returns 0 if the current connection is the last one.

for an example, see `Gpm_GetEvent` (498).

Errors: None.

See also: `Gpm_Open` (500)

gpm_fitvalues

Synopsis: Change coordinates to fit physical screen.

Declaration: `function gpm_fitvalues(var x: LongInt; var y: LongInt) : LongInt`

Visibility: default

Description: `Gpm_fitValues` changes `x` and `y` so they fit in the visible screen. The actual mouse pointer is not affected by this function.

Errors: None.

See also: `Gpm_FitValuesM` (498)

gpm_fitvaluesM

Synopsis: Change coordinates to fit margin.

Declaration: `function gpm_fitvaluesM(var x: LongInt; var y: LongInt; margin: LongInt) : LongInt`

Visibility: default

Description: `Gpm_FitValuesM` changes `x` and `y` so they fit in the margin indicated by `margin`. If `margin` is -1, then the values are fitted to the screen. The actual mouse pointer is not affected by this function.

Errors: None.

See also: `Gpm_FitValues` (498)

gpm_getevent

Synopsis: Get event from event queue.

Declaration: `function gpm_getevent(var event: Tgpm_event) : LongInt`

Visibility: default

Description: `Gpm_GetEvent` Reads an event from the file descriptor `gpm_fd`. This file is only for internal use and should never be called by a client application.

It returns 1 on succes, and -1 on failue.

Errors: On error, -1 is returned.

See also: `Gpm_GetSnapshot` (499)

Listing: ./gpmex/gpmex.pp

```

program gpmex;

{
  Example program to demonstrate the use of the gpm unit.
}

uses gpm;

var
  connect : TGPMConnect;
  event : tgpmevent;

begin
  connect.EventMask:=GPM_MOVE or GPM_DRAG or GPM_DOWN or GPM_UP;
  connect.DefaultMask:=0;
  connect.MinMod:=0;
  connect.MaxMod:=0;
  if Gpm_Open(connect,0)=-1 then
    begin
      WriteLn('No mouse handler present. ');
      Halt(1);
    end;
  WriteLn('Click right button to end. ');
  Repeat
    gpm_getevent(Event);
    With Event do
      begin
        Write('Pos = ( ',X,' ',Y,' ) Buttons : ( ');
        if (buttons and Gpm_b_left)<>0 then
          write('left ');
        if (buttons and Gpm_b_right)<>0 then
          write('right ');
        if (buttons and Gpm_b_middle)<>0 then
          Write('middle ');
        Write(') Event : ');
        Case EventType and $F of
          GPM_MOVE: write('Move ');
          GPM_DRAG: write('Drag ');
          GPM_DOWN: write('Down ');
          GPM_UP: write('Up ');
        end;
        WriteLn;
      end;
    Until (Event.Buttons and gpm_b_right)<>0;
    gpm_close;
  end.

```

gpm_getsnapshot

Synopsis: Return servers' current image of mouse state.

Declaration: function gpm_getsnapshot(eptr: Pgpmevent) : LongInt
 function gpm_getsnapshot(var eptr: Tgpmevent) : LongInt

Visibility: default

Description: `Gpm_GetSnapshot` returns the picture that the server has of the current situation in `Event`. This call will not read the current situation from the mouse file descriptor, but returns a buffered version. The function returns the number of mouse buttons, or -1 if this information is not available.

Errors: None.

See also: `Gpm_GetEvent` ([498](#))

gpm_lowerroi

Synopsis: Lower a region of interest in the stack.

Declaration: `function gpm_lowerroi(which: Pgpm_roi;after: Pgpm_roi) : Pgpm_roi`

Visibility: default

Description: `Gpm_LowerRoi` lowers the region of interest which after `after`. If `after` is `Nil`, the region of interest is moved to the bottom of the stack.

The return value is the new top of the region-of-interest stack.

Errors: None.

See also: `Gpm_RaiseRoi` ([501](#)), `Gpm_PopRoi` ([500](#)), `Gpm_PushRoi` ([501](#))

gpm_open

Synopsis: Open connection to GPM server.

Declaration: `function gpm_open(var conn: Tgpm_connect;flag: LongInt) : LongInt`

Visibility: default

Description: `Gpm_Open` opens a new connection to the mouse server. The connection is described by the fields of the `conn` record of type `TGPMConnect` ([495](#)).

if `Flag` is 0, then the application only receives events that come from its own terminal device. If it is negative it will receive all events. If the value is positive then it is considered a console number to which to connect.

The return value is -1 on error, or the file descriptor used to communicate with the client. Under an X-Term the return value is -2.

for an example, see `Gpm_GetEvent` ([498](#)).

Errors: On Error, the return value is -1.

See also: `Gpm_Open` ([500](#))

gpm_poproi

Synopsis: Pop region of interest from the stack.

Declaration: `function gpm_poproi(which: Pgpm_roi) : Pgpm_roi`

Visibility: default

Description: `Gpm_PopRoi` pops the topmost region of interest from the stack. It returns the next element on the stack, or `Nil` if the current element was the last one.

Errors: None.

See also: `Gpm_RaiseRoi` ([501](#)), `Gpm_LowerRoi` ([500](#)), `Gpm_PushRoi` ([501](#))

gpm_pushroi

Synopsis: Push region of interest on the stack.

Declaration: `function gpm_pushroi(x1: LongInt; y1: LongInt; x2: LongInt; y2: LongInt;
mask: LongInt; fun: TGpmHandler; xtradata: pointer)
: Pgpm_roi`

Visibility: default

Description: `Gpm_PushRoi` puts a new *region of interest* on the stack. The region of interest is defined by a rectangle described by the corners (X1, Y1) and (X2, Y2).

The mask describes which events the handler {fun} will handle; ExtraData will be put in the xtradata field of the {TGPM_Roi} record passed to the fun handler.

Errors: None.

See also: `Gpm_RaiseRoi` ([501](#)), `Gpm_PopRoi` ([500](#)), `Gpm_LowerRoi` ([500](#))

gpm_raiseroi

Synopsis: Raise region of interest in the stack.

Declaration: `function gpm_raiseroi(which: Pgpm_roi; before: Pgpm_roi) : Pgpm_roi`

Visibility: default

Description: `Gpm_RaiseRoi` raises the *region of interest* which till it is on top of region before. If before is nil then the region is put on top of the stack. The returned value is the top of the stack.

Errors: None.

See also: `Gpm_PushRoi` ([501](#)), `Gpm_PopRoi` ([500](#)), `Gpm_LowerRoi` ([500](#))

gpm_repeat

Synopsis: Check for presence of mouse event.

Declaration: `function gpm_repeat(millisecond: LongInt) : LongInt`

Visibility: default

Description: `Gpm_Repeat` returns 1 if no mouse event arrives in the next millisecond, it returns 0 otherwise.

Errors: None.

See also: `Gpm_GetEvent` ([498](#))

Gpm_StrictDouble

Synopsis: Check whether event contains only a double-click event.

Declaration: `function Gpm_StrictDouble(EventType: LongInt) : Boolean`

Visibility: default

Description: `Gpm_StrictDouble` returns true if EventType contains only a doubleclick event, False otherwise.

Errors: None.

See also: [Gpm_StrictSingle \(502\)](#), [Gpm_AnyTriple \(497\)](#), [Gpm_AnyDouble \(497\)](#), [Gpm_StrictTriple \(502\)](#), [Gpm_AnySingle \(497\)](#)

Gpm_StrictSingle

Synopsis: Check whether event contains only a single-click event.

Declaration: `function Gpm_StrictSingle(EventType: LongInt) : Boolean`

Visibility: default

Description: `Gpm_StrictDouble` returns `True` if `EventType` contains only a singleclick event, `False` otherwise.

Errors: None.

See also: [Gpm_AnyTriple \(497\)](#), [Gpm_StrictDouble \(501\)](#), [Gpm_AnyDouble \(497\)](#), [Gpm_StrictTriple \(502\)](#), [Gpm_AnySingle \(497\)](#)

Gpm_StrictTriple

Synopsis: Check whether event contains only a triple-click event.

Declaration: `function Gpm_StrictTriple(EventType: LongInt) : Boolean`

Visibility: default

Description: `Gpm_StrictTriple` returns `true` if `EventType` contains only a triple click event, `False` otherwise.

Errors: None.

See also: [Gpm_AnyTriple \(497\)](#), [Gpm_StrictDouble \(501\)](#), [Gpm_AnyDouble \(497\)](#), [Gpm_StrictSingle \(502\)](#), [Gpm_AnySingle \(497\)](#)

Chapter 12

Reference for unit 'Graph'

12.1 Categorized functions: Text and font handling

Functions to set texts on the screen.

Table 12.1:

Name	Description
GetTextSettings (537)	Get current text settings
InstallUserFont (540)	Install a new font
OutText (541)	Write text at current cursor position
OutTextXY (528)	Write text at coordinates X,Y
RegisterBGIFont (542)	Register a new font
SetTextJustify (546)	Set text justification
SetTextStyle (546)	Set text style
SetUserCharSize (547)	Set text size
TextHeight (548)	Calculate height of text
TextWidth (548)	Calculate width of text

12.2 Categorized functions: Filled drawings

Functions for drawing filled regions.

12.3 Categorized functions: Drawing primitives

Functions for simple drawing.

12.4 Categorized functions: Color management

All functions related to color management.

Table 12.2:

Name	Description
Bar3D (530)	Draw a filled 3D-style bar
Bar (529)	Draw a filled rectangle
FloodFill (532)	Fill starting from coordinate
FillEllipse (531)	Draw a filled ellipse
FillPoly (531)	Draw a filled polygone
GetFillPattern (534)	Get current fill pattern
GetFillSettings (534)	Get current fill settings
SetFillPattern (544)	Set current fill pattern
SetFillStyle (544)	Set current fill settings

Table 12.3:

Name	Description
Arc (529)	Draw an arc
Circle (527)	Draw a complete circle
DrawPoly (531)	Draw a polygone with N points
Ellipse (531)	Draw an ellipse
GetArcCoords (532)	Get arc coordinates
GetLineSettings (534)	Get current line drawing settings
Line (528)	Draw line between 2 points
LineRel (540)	Draw line relative to current position
LineTo (540)	Draw line from current position to absolute position
MoveRel (540)	Move cursor relative to current position
MoveTo (541)	Move cursor to absolute position
PieSlice (541)	Draw a pie slice
PutPixel (528)	Draw 1 pixel
Rectangle (542)	Draw a non-filled rectangle
Sector (543)	Draw a sector
SetLineStyle (545)	Set current line drawing style

12.5 Categorized functions: Screen management

General drawing screen management functions.

12.6 Categorized functions: Initialization

Initialization of the graphics screen.

12.7 Target specific issues: Linux

There are several issues on Linux that need to be taken care of:

The Linux version of the Graph unit uses the libvga library. This library works on the console, not under X.

If you get an error similar to

Table 12.4:

Name	Description
GetBkColor (532)	Get current background color
GetColor (533)	Get current foreground color
GetDefaultPalette (533)	Get default palette entries
GetMaxColor (535)	Get maximum valid color
GetPaletteSize (536)	Get size of palette for current mode
GetPixel (527)	Get color of selected pixel
GetPalette (536)	Get palette entry
SetAllPalette (529)	Set all colors in palette
SetBkColor (543)	Set background color
SetColor (543)	Set foreground color
SetPalette (545)	Set palette entry
SetRGBPalette (529)	Set palette entry with RGB values

Table 12.5:

Name	Description
ClearViewPort (527)	Clear the current viewport
GetImage (527)	Copy image from screen to memory
GetMaxX (535)	Get maximum X coordinate
GetMaxY (535)	Get maximum Y coordinate
GetX (537)	Get current X position
GetY (537)	Get current Y position
ImageSize (528)	Get size of selected image
GetViewSettings (537)	Get current viewport settings
PutImage (528)	Copy image from memory to screen
SetActivePage (529)	Set active video page
SetAspectRatio (543)	Set aspect ratio for drawing routines
SetViewPort (547)	Set current viewport
SetVisualPage (529)	Set visual page
SetWriteMode (547)	Set write mode for screen operations

```
/usr/bin/ld: cannot find -lvga
```

This can mean one of two things: either libvga and its development package is not installed properly, or the directory where it is installed is not in the linker path.

To remedy the former, you should install both the libvga package and libvga-devel package (or compile and install from scratch).

To remedy the latter, you should add the path to the compiler command-line using the `-F1` option.

Programs using **libvga** need root privileges to run. You can make them `setuid` root with the following command:

```
chown root.root myprogram
chmod u+s myprogram
```

The libvga library will give up the root privileges after it is initialized.

there is an experimental version of the Graphics library available that uses GGI to do all the drawing, but it is not well tested. It's called **ggigraph** and is distributed in source form only.

Table 12.6:

Name	Description
<code>ClearDevice</code> (530)	Empty the graphics screen
<code>CloseGraph</code> (530)	Finish drawing session, return to text mode
<code>DetectGraph</code> (530)	Detect graphical modes
<code>GetAspectRatio</code> (532)	Get aspect ratio of screen
<code>GetModeRange</code> (536)	Get range of valid modes for current driver
<code>GraphDefaults</code> (538)	Set defaults
<code>GetDriverName</code> (533)	Return name of graphical driver
<code>GetGraphMode</code> (534)	Return current or last used graphics mode
<code>GetMaxMode</code> (535)	Get maximum mode for current driver
<code>GetModeName</code> (536)	Get name of current mode
<code>GraphErrorMsg</code> (538)	String representation of graphical error
<code>GraphResult</code> (538)	Result of last drawing operation
<code>InitGraph</code> (539)	Initialize graphics drivers
<code>InstallUserDriver</code> (539)	Install a new driver
<code>RegisterBGIDriver</code> (542)	Register a new driver
<code>RestoreCRTMode</code> (542)	Go back to text mode
<code>SetGraphMode</code> (545)	Set graphical mode

Do not use the CRT unit together with the Graph unit: the console may end up in an unusable state. Instead, the `ncurses` unit may function fine.

12.8 Target specific issues: DOS

VESA modes (i.e., anything but 320x200x256 and 640x480x16) do not work under most installations of Windows NT, Windows 2000 and Windows XP. They also do not work for some people under Windows 98 and Windows ME, depending on their graphics drivers. However, the graph unit cannot detect this, because no errors are returned from the system. In such cases, the screen simply turns black, or will show garbage.

Nothing can be done about this, the reason is missing or buggy support in the graphics drivers of the operating system.

12.9 A word about mode selection

The graph unit was implemented for compatibility with the old Turbo Pascal graph unit. For this reason, the mode constants as they were defined in the Turbo Pascal graph unit are retained.

However, since

1. Video cards have evolved very much
2. Free Pascal runs on multiple platforms

it was decided to implement new mode and graphic driver constants, which are more independent of the specific platform the program runs on.

In this section we give a short explanation of the new mode system. the following drivers were defined:

```

D1bit = 11;
D2bit = 12;
D4bit = 13;
D6bit = 14; { 64 colors Half-brite mode - Amiga }
D8bit = 15;
D12bit = 16; { 4096 color modes HAM mode - Amiga }
D15bit = 17;
D16bit = 18;
D24bit = 19; { not yet supported }
D32bit = 20; { not yet supported }
D64bit = 21; { not yet supported }

lowNewDriver = 11;
highNewDriver = 21;

```

Each of these drivers specifies a desired color-depth.

The following modes have been defined:

```

detectMode = 30000;
m320x200 = 30001;
m320x256 = 30002; { amiga resolution (PAL) }
m320x400 = 30003; { amiga/atari resolution }
m512x384 = 30004; { mac resolution }
m640x200 = 30005; { vga resolution }
m640x256 = 30006; { amiga resolution (PAL) }
m640x350 = 30007; { vga resolution }
m640x400 = 30008;
m640x480 = 30009;
m800x600 = 30010;
m832x624 = 30011; { mac resolution }
m1024x768 = 30012;
m1280x1024 = 30013;
m1600x1200 = 30014;
m2048x1536 = 30015;

lowNewMode = 30001;
highNewMode = 30015;

```

These modes start at 30000 because Borland specified that the mode number should be ascending with increasing X resolution, and the new constants shouldn't interfere with the old ones.

The above constants can be used to set a certain color depth and resolution, as demonstrated in the below example.

If other modes than the ones above are supported by the graphics card, you will not be able to select them with this mechanism.

For this reason, there is also a 'dynamic' mode number, which is assigned at run-time. This number increases with increasing X resolution. It can be queried with the `getmoderange` call. This call will return the range of modes which are valid for a certain graphics driver. The numbers are guaranteed to be consecutive, and can be used to search for a certain resolution, as in the second example below.

Thus, the `getmoderange` function can be used to detect all available modes and drivers, as in the third example below:

12.10 Requirements

The unit Graph exports functions and procedures for graphical output. It requires at least a VGA-compatible Card or a VGA-Card with software-driver (min. **512Kb** video memory).

12.11 Overview

This document describes the GRAPH unit for Free Pascal, for all platforms. The unit was first written for dos by Florian kl\"ampfl, but was later completely rewritten by Carl-Eric Codere to be completely portable. The unit is provided for compatibility only: It is recommended to use more modern graphical systems. The graph unit will allow to recompile old programs, they will work to some extent, but if the application has heavy graphical needs, it's recommended to use another set of graphical routines, suited to the platform the program should work on.

12.12 Constants, types and variables

Constants

AndPut = 3

Draw operation: use AND

```
AnsiToASCIITransTable : TCharsetTransTable = (#$00,$$01,$$02,$$03,$$04,$$05,$$06,$$07,
```

Default ansi transliteration table.

BkSlashFill = 5

Fill style: Diagonal (backslash) lines

```
black = 0
```

Color code: black.

```
blue = 1
```

Color code: blue

BoldFont = 10

Font number: Bold font.

BottomText = 0

Vertical text alignment: Align text to bottom

brown = 6

Color code: brown

CenterLn = 2

Line style: centered line

CenterText = 1

Horizontal text alignment: Center text

ClipOff = false

Viewport clipping off

ClipOn = true

Viewport clipping on

CloseDotFill = 11

Fill style: Closely spaced dotted lines

CopyPut = 0

Draw operation: use Copy

CurrentDriver = -128

Currently used driver

cyan = 3

Color code: Cyan

D12bit = 16

Mode: Depth 12 bit

D15bit = 17

Mode: Depth 15 bit

D16bit = 18

Mode: Depth 16 bit

D1bit = 11

Mode: Depth 1 bit

D24bit = 19

Mode: Depth 24 bit

D2bit = 12

Mode: Depth 2 bit

D32bit = 20

Mode: Depth 32 bit

D4bit = 13

Mode: Depth 4 bit

D64bit = 21

Mode: Depth 64 bit

D6bit = 14

Mode: Depth 6 bit

D8bit = 15

Mode: Depth 8 bit

darkgray = 8

Color code: Dark gray

DashedLn = 3

Line style: dashed line

Default = 0

Default mode

DefaultFont = 0

Font number: Normal font

Detect = 0

Mode: Detect mode.

detectMode = 30000

Mode: Autodetect optimal mode

DottedLn = 1

Line style: Dotted line

DrawTextBackground : Boolean = false

Should the background of texts be drawn or should it be left untouched ?

EGABlack = 0

Color code: EGA Black

EGABlue = 1

Color code: EGA blue

EGABrown = 20

Color code: EGA brown

EGACyan = 3

Color code: EGA cyan

EGADarkgray = 56

Color code: EGA dark gray

EGAGreen = 2

Color code: EGA green

EGALightblue = 57

Color code: EGA Light blue

EGALightcyan = 59

Color code: EGA Light cyan

EGALightgray = 7

Color code: EGA Light gray

EGALightgreen = 58

Color code: EGA Light green

EGALightmagenta = 61

Color code: EGA light magenta

EGALightred = 60

Color code: EGA light red

EGAMagenta = 5

Color code: EGA magenta

EGARed = 4

Color code: EGA red

EGAWhite = 63

Color code: EGA white

EGAYellow = 62

Mode: Resolution 1152x864, 32K colors

G1152x864x64K = 41

Mode: Resolution 1152x864, 64K colors

G1280x1024x16 = 31

Mode: Resolution 1280x1024, 16 colors

G1280x1024x16M = 28

Mode: Resolution 1280x1024, 16M colors

G1280x1024x16M32 = 37

Mode: Resolution 1280x1024, 16M 32-bit colors

G1280x1024x256 = 13

Mode: Resolution 1280x1024, 256 colors

G1280x1024x32K = 26

Mode: Resolution 1280x1024, 32K colors

G1280x1024x64K = 27

Mode: Resolution 1280x1024, 64K colors

G1600x1200x16 = 44

Mode: Resolution 1600x1200, 16 colors

G1600x1200x16M = 48

Mode: Resolution 1600x1200, 16M colors

G1600x1200x16M32 = 49

Mode: Resolution 1600x1200, 16M 32-bit colors

G1600x1200x256 = 45

Mode: Resolution 1600x1200, 256 colors

G1600x1200x32K = 46

Mode: Resolution 1600x1200, 32K colors

G1600x1200x64K = 47

Mode: Resolution 1600x1200, 64K colors

G320x200x16 = 1

Mode: Resolution 320x200, 16 colors

G320x200x16M = 16

Mode: Resolution 320x200, 16M colors

G320x200x16M32 = 33

Mode: Resolution 320x200, 16M 32-bit colors

G320x200x256 = 5

Mode: Resolution 320x200, 256 colors

G320x200x32K = 14

Mode: Resolution 320x200, 32K colors

G320x200x64K = 15

Mode: Resolution 320x200, 64K colors

G320x240x256 = 6

Mode: Resolution 320x240, 256 colors

G320x400x256 = 7

Mode: Resolution 320x400, 256 colors

G360x480x256 = 8

Mode: Resolution 360x480, 256 colors

G640x200x16 = 2

Mode: Resolution x, colors

G640x350x16 = 3

Mode: Resolution x, colors

G640x480x16 = 4

Mode: Resolution x, colors

G640x480x16M = 19

Mode: Resolution 640x480, 16M colors

G640x480x16M32 = 34

Mode: Resolution 640x480, 16M 32-bit colors

G640x480x2 = 9

Mode: Resolution 640x480, 2 colors

G640x480x256 = 10

Mode: Resolution 640x480, 256 colors

G640x480x32K = 17

Mode: Resolution 640x480, 32K colors

G640x480x64K = 18

Mode: Resolution 640x480, 64K colors

G720x348x2 = 32

Mode: Resolution 720x348, 2 colors

G800x600x16 = 29

Mode: Resolution 800x600, 16 colors

G800x600x16M = 22

Mode: Resolution 800x600, 16M colors

G800x600x16M32 = 35

Mode: Resolution 800x600, 16M 32-bit colors

G800x600x256 = 11

Mode: Resolution 800x600, 256 colors

G800x600x32K = 20

Mode: Resolution 800x600, 32K colors

G800x600x64K = 21

Mode: Resolution 800x600, 64K colors

GothicFont = 4

Font number: Gothic font

GraphStringTransTable : PCharsetTransTable = nil

Table used when transliterating strings.

green = 2

Color code: green

grError = -11

Error: Unknown error.

grFileNotFound = -3

Error: File for driver not found.

grFontNotFound = -8

Error: font description file not found.

grInvalidDriver = -4

Error: Invalid driver specified

grInvalidFont = -13

Error: Invalid font description

grInvalidFontNum = -14

Error: Invalid font number

grInvalidMode = -10

Error: Invalid mode specified.

grInvalidVersion = -18

Error: Invalid version.

grIOerror = -12

Error: Unspecified Input/Output error.

grNoFloodMem = -7

Error: Could not allocate memory for flood operation.

grNoFontMem = -9

Error: Not enough memory to load font.

grNoInitGraph = -1

Error: Graphical system not initialized

grNoLoadMem = -5

Error: Memory error.

grNoScanMem = -6

Error: Could not allocate memory for scan

grNotDetected = -2

Error: Graphics device not detected.

grOk = 0

Graphical operation went OK.

HatchFill = 7

Fill style: Hatch lines

HercMono = 7

Mode: Hercules, mono color

HercMonoHi = 0

Mode: Hercules card, monochrome, high resolution

highNewDriver = 21

Mode: highest number for new driver

highNewMode = 30015

Mode: Highest possible value of the new modes.

HorizDir = 0

Text write direction: Horizontal

InterleaveFill = 9

Fill style: Interleaving lines

LCOMFont = 8

Font number: ?

LeftText = 0

Horizontal text alignment: Align text left

lightblue = 9

Color code: Light blue

lightcyan = 11

Color code: Light cyan

lightgray = 7

Color code: Light gray

lightgreen = 10

Color code: Light green

`lightmagenta = 13`

Color code: Light magenta

`lightred = 12`

Color code: Light red

`LineFill = 2`

Fill style: Fill using horizontal lines

`lowNewDriver = 11`

Mode: lowest number for new driver

`lowNewMode = 30001`

Mode: Lowest possible value of the new modes.

`LowRes = 1`

Mode: Low resolution.

`LtBkSlashFill = 6`

Fill style: Light diagonal (backslash) lines

`LtSlashFill = 3`

Fill style: Light diagonal (slash) lines

`m1024x768 = 30012`

Mode: Resolution 1024x768

`m1280x1024 = 30013`

Mode: Resolution 1280x1024

`m1600x1200 = 30014`

Mode: Resolution 1600x1200

`m2048x1536 = 30015`

Mode: Resolution 2048x1536

`m320x200 = 30001`

Mode: Resolution 320x200

`m320x256 = 30002`

Mode: Resolution 320x256

m320x400 = 30003

Mode: Resolution 320x400

m512x384 = 30004

Mode: Resolution 512x384

m640x200 = 30005

Mode: Resolution 640x200

m640x256 = 30006

Mode: Resolution 640x256

m640x350 = 30007

Mode: Resolution 640x350

m640x400 = 30008

Mode: Resolution 640x400

m640x480 = 30009

Mode: Resolution 640x480

m800x600 = 30010

Mode: Resolution 800x600

m832x624 = 30011

Mode: Resolution 832x624

magenta = 5

Color code: Magenta

MaxColors = 255

Max amount of colors in a palette

maxsmallint = high (smallint)

Maximum value for smallint type

NormalPut = 0

Draw operation: Use Normal (copy) operation

NormWidth = 1

Line width: Normal width

NotPut = 4

Draw operation: use NOT

OrPut = 2

Draw operation: use OR

red = 4

Color code: Red

resolutions : Array[lowNewMode..highNewMode] of TResolutionRec = ((x:320;y:200) ,

Array with actual resolutions of the new modes

RightText = 2

Horizontal text alignment: Align text right

SansSerifFont = 3

Font number: Sans Serif font

ScriptFont = 5

Font number: Script font

SimpleFont = 6

Font number: Simple font

SlashFill = 4

Fill style: Diagonal (slash) lines

SmallFont = 2

Font number: Small font

SolidFill = 1

Fill style: Solid fill.

SolidLn = 0

Line style: Solid line

ThickWidth = 3

Line width: double width

TopOff = false

Top off

TopOn = true

Top on

TopText = 2

Vertical text alignment: Align text to top

TriplexFont = 1

Font number: Triplex font

TSCRFont = 7

Font number: Terminal font

UserBitLn = 4

Line style: User defined

UserCharSize = 0

User character size

UserFill = 12

Fill style: User-defined fill.

VertDir = 1

Text write direction: Vertical

VESA = 10

Mode: VESA graphics adaptor.

VGA = 9

Mode: VGA graphics adaptor.

VGAHi = 2

Mode: VGA high resolution (640x480)

VGALo = 0

Mode: VGA low resolution (640x200)

VGAMed = 1

Mode: VGA medium resolution (640x350)

white = 15

Color code: White

WideDotFill = 10

Fill style: Widely spaced dotted lines

XHatchFill = 8

Fill style: Heavy hatch lines

XORPut = 1

Draw operation: use XOR

yellow = 14

Color code: Yellow

Types

```
ArcCoordsType = record
  x : SmallInt;
  y : SmallInt;
  xstart : SmallInt;
  ystart : SmallInt;
  xend : SmallInt;
  yend : SmallInt;
end
```

Describe the last arc which was drawn on screen

```
CircleProc = procedure(X: SmallInt;Y: SmallInt;Radius: Word)
```

Standard circle drawing routine prototype.

```
clrviewproc = procedure
```

Standard clearviewport routine prototype

```
defpixelproc = procedure(X: SmallInt;Y: SmallInt)
```

This is the standard putpixel routine used by all function drawing routines, it will use the viewport settings, as well as clip, and use the current foreground color to plot the desired pixel.

```
ellipseproc = procedure(X: SmallInt;Y: SmallInt;XRadius: Word;
                        YRadius: Word;stAngle: Word;EndAngle: Word;
                        fp: patternlineproc)
```

Standard ellipse drawing routine prototype.

```
FillPatternType = Array[1..8] of Byte
```

Bit pattern used when drawing lines. Set bits are drawn.

```
FillSettingsType = record
  pattern : Word;
  color : Word;
end
```

Record describing fill mode

```
getimageproc = procedure(X1: SmallInt;Y1: SmallInt;X2: SmallInt;
                          Y2: SmallInt;var Bitmap)
```

Standard GetImage (527) procedure prototype.

```
getpixelproc = function(X: SmallInt;Y: SmallInt) : Word
```

Standard pixel fetching routine prototype

```
getrgbpaletteproc = procedure(ColorNum: SmallInt;var RedValue: SmallInt;
                              var GreenValue: SmallInt;
                              var BlueValue: SmallInt)
```

This routine prototype is a hook for GetRGBPalette (527)

```
getscanlineproc = procedure(X1: SmallInt;X2: SmallInt;Y: SmallInt;
                             var data)
```

This routine is used for FloodFill (532) It returns an entire screen scan line with a word for each pixel in the scanline. Also handy for GetImage.

```
graphfreememprc = procedure(var P: Pointer;size: Word)
```

Procedure prototype, used when heap memory is freed by the graph routines.

```
graphgetmemprc = procedure(var P: pointer;size: Word)
```

Procedure prototype, used when heap memory is needed by the graph routines.

```
graph_float = single
```

The platform's preferred floating point size for fast graph operations

```
hlineproc = procedure(x: SmallInt;x2: SmallInt;y: SmallInt)
```

Standard procedure prototype to draw a single horizontal line

```
imagesizeproc = function(X1: SmallInt;Y1: SmallInt;X2: SmallInt;
                          Y2: SmallInt) : LongInt
```

Standard ImageSize (528) calculation procedure prototype.

```
initmodeproc = procedure
```

Standard routine prototype to initialize a mode.


```
lineproc = procedure(X1: SmallInt;Y1: SmallInt;X2: SmallInt;
                    Y2: SmallInt)
```

Standard line drawing routine prototype.

```
LineSettingsType = record
    linestyle : Word;
    pattern : Word;
    thickness : Word;
end
```

Record describing current line drawing mode

```
OutTextXYProc = procedure(x: SmallInt;y: SmallInt;
                        const TextString: String)
```

This routine prototype is a hook for OutTextXY ([528](#))

```
PaletteType = record
    Size : LongInt;
    Colors : Array[0..MaxColors] of RGBRec;
end
```

Record describing palette.

```
patternlineproc = procedure(x1: SmallInt;x2: SmallInt;y: SmallInt)
```

Standard procedure prototype to draw a patterned line

```
PCharsetTransTable = ^TCharsetTransTable
```

Pointer to TCharsetTransTable ([525](#)) array.

```
PModeInfo = ^TModeInfo
```

Pointer to TModeInfo ([526](#)) record

```
PointType = record
    x : SmallInt;
    y : SmallInt;
end
```

Record describing a point in a 2 dimensional plane

```
putimageproc = procedure(X: SmallInt;Y: SmallInt;var Bitmap;
                        BitBlt: Word)
```

Standard PutImage ([528](#)) procedure prototype.

```
putpixelproc = procedure(X: SmallInt;Y: SmallInt;Color: Word)
```

Standard pixel drawing routine prototype

```
restorestateproc = procedure
```

Standard routine prototype to restore the graphical state at a closegraph call.

```
RGBRec = packed record
  Red : SmallInt;
  Green : SmallInt;
  Blue : SmallInt;
end
```

Record describing palette RGB color

```
savestateproc = procedure
```

Standard routine prototype to save the graphical state before a mode is set.

```
setactivepageproc = procedure(page: Word)
```

Standard SetActivePage (529) procedure prototype.

```
SetAllPaletteProc = procedure(const Palette: PaletteType)
```

This routine prototype is a hook for SetAllPalette (529)

```
setrgbpaletteproc = procedure(ColorNum: SmallInt; RedValue: SmallInt;
                               GreenValue: SmallInt; BlueValue: SmallInt)
```

This routine prototype is a hook for SetRGBPalette (529)

```
setvisualpageproc = procedure(page: Word)
```

Standard SetVisualPage (529) procedure prototype.

```
smallint = -32768..32767
```

Type redefinition

```
TCharsetTransTable = Array[Char] of Char
```

Character transliteration table, with entries for 256 characters

```
TextSettingsType = record
  font : Word;
  direction : Word;
  charsize : Word;
  horiz : Word;
  vert : Word;
end
```

Record describing how texts are drawn.

```

TModeInfo = record
  DriverNumber : SmallInt;
  ModeNumber : SmallInt;
  internModeNumber : SmallInt;
  MaxColor : LongInt;
  PaletteSize : LongInt;
  XAspect : Word;
  YAspect : Word;
  MaxX : Word;
  MaxY : Word;
  DirectColor : Boolean;
  Hardwarepages : Byte;
  ModeName : String;
  DirectPutPixel : defpixelproc;
  GetPixel : getpixelproc;
  PutPixel : putpixelproc;
  SetRGBPalette : setrgbpaletteproc;
  GetRGBPalette : getrgbpaletteproc;
  SetAllPalette : SetAllPaletteProc;
  SetVisualPage : setvisualpageproc;
  SetActivePage : setactivepageproc;
  ClearViewPort : clrviewproc;
  PutImage : putimageproc;
  GetImage : getimageproc;
  ImageSize : imagesizeproc;
  GetScanLine : getscanlineproc;
  Line : lineproc;
  InternalEllipse : ellipseproc;
  PatternLine : patternlineproc;
  HLine : hlineproc;
  VLine : vlineproc;
  Circle : CircleProc;
  InitMode : initmodeproc;
  OutTextXY : OutTextXYProc;
  next : PModeInfo;
end

```

Record describing a graphical mode.

```

TNewModeInfo = record
  modeInfo : Array[lowNewDriver..highNewDriver] of PModeInfo;
  loHiModeNr : Array[lowNewDriver..highNewDriver] of ;
end

```

Mode information for new modes.a

```

TResolutionRec = record
  x : LongInt;
  y : LongInt;
end

```

Record describing resolution

```
ViewPortType = record
  x1 : SmallInt;
  y1 : SmallInt;
  x2 : SmallInt;
  y2 : SmallInt;
  Clip : Boolean;
end
```

Record describing a viewport

```
vlviewproc = procedure(x: SmallInt;y: SmallInt;y2: SmallInt)
```

Standard procedure prototype to draw a single vertical line

Variables

```
Circle : CircleProc
```

Circle draws a complete circle with center at (X,Y), radius radius.

```
ClearViewPort : clrviewproc
```

Clears the current viewport. The current background color is used as filling color. The pointer is set at (0,0).

```
DirectPutPixel : defpixelproc
```

Hook to directly draw a pixel on the screen.

```
GetImage : getimageproc
```

GetImage Places a copy of the screen area (X1,Y1) to X2,Y2 in BitMap

```
GetPixel : getpixelproc
```

GetPixel returns the color of the point at (X,Y)

```
GetRGBPalette : getrgbpaletteproc
```

Hook to set a RGB palette entries.

```
GetScanLine : getscanlineproc
```

Hook to get a scan line from the screen.

```
GraphFreeMemPtr : graphfreememprc
```

Hook to free heap memory.

```
GraphGetMemPtr : graphgetmemprc
```

Hook to get heap memory

HLine : hlineproc

Hook to draw a solid horizontal line

ImageSize : imagesizeproc

ImageSize returns the number of bytes needed to store the image in the rectangle defined by (X1,Y1) and (X2,Y2).

InternalEllipse : ellipseproc

Hook to draw an ellipse

Line : lineproc

Line draws a line starting from (X1,Y1 to (X2,Y2), in the current line style and color. The current position is put to (X2,Y2)

OutTextXY : OutTextXYProc

OutText puts TextString on the screen, at position (X,Y), using the current font and text settings. The current position is moved to the end of the text.

PatternLine : patternlineproc

Hook to draw a patterned line

PutImage : putimageproc

PutImage Places the bitmap in Bitmap on the screen at (X1,Y1). How determines how the bitmap will be placed on the screen. Possible values are :

- CopyPut
- XORPut
- ORPut
- AndPut
- NotPut

PutPixel : putpixelproc

Puts a point at (X,Y) using color Color

RestoreVideoState : restorestateproc

Hook to restore a saved video mode

SaveVideoState : savestateproc

Hook to save the current video state

SetActivePage : setactivepageproc

Sets Page as the active page for all graphical output.

`SetAllPalette : SetAllPaletteProc`

Sets the current palette to `Palette`. `Palette` is an untyped variable, usually pointing to a record of type `PaletteType`

`SetRGBPalette : setrgbpaletteproc`

`SetRGBPalette` sets the `ColorNr`-th entry in the palette to the color with RGB-values `Red`, `Green` `Blue`.

`SetVisualPage : setvisualpageproc`

`SetVisualPage` sets the video page to page number `Page`.

`VLine : vlineproc`

Hook to draw a solid vertical line

12.13 Procedures and functions

Arc

Synopsis: Draw part of a circle

Declaration: `procedure Arc(X: SmallInt;Y: SmallInt;StAngle: Word;EndAngle: Word;
Radius: Word)`

Visibility: default

Description: `Arc` draws part of a circle with center at `(X,Y)`, radius `radius`, starting from angle `start`, stopping at angle `stop`. These angles are measured counterclockwise.

Errors: None.

See also: `Circle` ([527](#)), `Ellipse` ([531](#)), `GetArcCoords` ([532](#)), `PieSlice` ([541](#)), `Sector` ([543](#))

Bar

Synopsis: Draw filled rectangle

Declaration: `procedure Bar(x1: SmallInt;y1: SmallInt;x2: SmallInt;y2: SmallInt)`

Visibility: default

Description: Draws a rectangle with corners at `(X1,Y1)` and `(X2,Y2)` and fills it with the current color and fill-style.

Errors: None.

See also: `Bar3D` ([530](#)), `Rectangle` ([542](#))

Bar3D

Synopsis: Draw filled 3-dimensional rectangle

Declaration: `procedure Bar3D(x1: SmallInt; y1: SmallInt; x2: SmallInt; y2: SmallInt;
depth: Word; top: Boolean)`

Visibility: default

Description: `Bar3d` draws a 3-dimensional Bar with corners at (X1,Y1) and (X2,Y2) and fills it with the current color and fill-style. Depth specifies the number of pixels used to show the depth of the bar.

If Top is true; then a 3-dimensional top is drawn.

Errors: None.

See also: Bar ([529](#)), Rectangle ([542](#))

ClearDevice

Synopsis: Clear the complete screen

Declaration: `procedure ClearDevice`

Visibility: default

Description: Clears the graphical screen (with the current background color), and sets the pointer at (0 , 0).

Errors: None.

See also: `ClearViewPort` ([527](#)), `SetBkColor` ([543](#))

Closegraph

Synopsis: Close graphical system.

Declaration: `procedure Closegraph`

Visibility: default

Description: Closes the graphical system, and restores the screen modus which was active before the graphical modus was activated.

Errors: None.

See also: `InitGraph` ([539](#))

DetectGraph

Synopsis: Detect correct graphical driver to use

Declaration: `procedure DetectGraph(var GraphDriver: SmallInt; var GraphMode: SmallInt)`

Visibility: default

Description: `DetectGraph` checks the hardware in the PC and determines the driver and screen-modus to be used. These are returned in `Driver` and `Modus`, and can be fed to `InitGraph`. See the `InitGraph` for a list of drivers and modi.

Errors: None.

See also: `InitGraph` ([539](#))

DrawPoly

Synopsis: Draw a polygone

Declaration: `procedure DrawPoly(NumPoints: Word; var polypoints)`

Visibility: default

Description: `DrawPoly` draws a polygone with `NumberOfPoints` corner points, using the current color and line-style. `PolyPoints` is an array of type `PointType` (524).

Errors: None.

See also: `Bar` (529), `Bar3D` (530), `Rectangle` (542)

Ellipse

Synopsis: Draw an ellipse

Declaration: `procedure Ellipse(X: SmallInt; Y: SmallInt; stAngle: Word; EndAngle: Word; XRadius: Word; YRadius: Word)`

Visibility: default

Description: `Ellipse` draws part of an ellipse with center at (X, Y) . `XRadius` and `Yradius` are the horizontal and vertical radii of the ellipse. `Start` and `Stop` are the starting and stopping angles of the part of the ellipse. They are measured counterclockwise from the X-axis (3 o'clock is equal to 0 degrees). Only positive angles can be specified.

Errors: None.

See also: `Arc` (529), `Circle` (527), `FillEllipse` (531)

FillEllipse

Synopsis: Draw and fill an ellipse

Declaration: `procedure FillEllipse(X: SmallInt; Y: SmallInt; XRadius: Word; YRadius: Word)`

Visibility: default

Description: `Ellipse` draws an ellipse with center at (X, Y) . `XRadius` and `Yradius` are the horizontal and vertical radii of the ellipse. The ellipse is filled with the current color and fill-style.

Errors: None.

See also: `Arc` (529), `Circle` (527), `GetArcCoords` (532), `PieSlice` (541), `Sector` (543)

FillPoly

Synopsis: Draw, close and fill a polygone

Declaration: `procedure FillPoly(NumPoints: Word; var PolyPoints)`

Visibility: default

Description: `FillPoly` draws a polygone with `NumberOfPoints` corner points and fills it using the current color and line-style. `PolyPoints` is an array of type `PointType`.

Errors: None.

See also: Bar ([529](#)), Bar3D ([530](#)), Rectangle ([542](#))

FloodFill

Synopsis: Fill an area with a given color

Declaration: `procedure FloodFill(x: SmallInt;y: SmallInt;Border: Word)`

Visibility: default

Description: Fills the area containing the point (X,Y), bounded by the color BorderColor.

Errors: None

See also: SetColor ([543](#)), SetBkColor ([543](#))

GetArcCoords

Synopsis: Return coordinates of last drawn arc or ellipse.

Declaration: `procedure GetArcCoords(var ArcCoords: ArcCoordsType)`

Visibility: default

Description: GetArcCoords returns the coordinates of the latest Arc or Ellipse call.

Errors: None.

See also: Arc ([529](#)), Ellipse ([531](#))

GetAspectRatio

Synopsis: Return screen resolution

Declaration: `procedure GetAspectRatio(var Xasp: Word;var Yasp: Word)`

Visibility: default

Description: GetAspectRatio determines the effective resolution of the screen. The aspect ration can then be calculated as $Xasp/Yasp$.

Errors: None.

See also: InitGraph ([539](#)), SetAspectRatio ([543](#))

GetBkColor

Synopsis: Return current background color

Declaration: `function GetBkColor : Word`

Visibility: default

Description: GetBkColor returns the current background color (the palette entry).

Errors: None.

See also: GetColor ([533](#)), SetBkColor ([543](#))

GetColor

Synopsis: Return current drawing color

Declaration: `function GetColor : Word`

Visibility: default

Description: `GetColor` returns the current drawing color (the palette entry).

Errors: None.

See also: `GetColor` ([533](#)), `SetBkColor` ([543](#))

GetDefaultPalette

Synopsis: Return default palette

Declaration: `procedure GetDefaultPalette(var Palette: PaletteType)`

Visibility: default

Description: `GetDefaultPalette` returns the current palette in `Palette`.

Errors: None.

See also: `GetColor` ([533](#)), `GetBkColor` ([532](#))

GetDirectVideo

Synopsis: Determine whether direct video mode is active.

Declaration: `function GetDirectVideo : Boolean`

Visibility: default

Description: Determine whether direct video mode is active.

Errors:

GetDriverName

Synopsis: Return current driver name

Declaration: `function GetDriverName : String`

Visibility: default

Description: `GetDriverName` returns a string containing the name of the current driver.

Errors: None.

See also: `GetModeName` ([536](#)), `InitGraph` ([539](#))

GetFillPattern

Synopsis: Return current fill pattern

Declaration: `procedure GetFillPattern(var FillPattern: FillPatternType)`

Visibility: default

Description: `GetFillPattern` returns an array with the current fill-pattern in `FillPattern`

Errors: None

See also: `SetFillPattern` ([544](#))

GetFillSettings

Synopsis: Return current fill settings

Declaration: `procedure GetFillSettings(var Fillinfo: FillSettingsType)`

Visibility: default

Description: `GetFillSettings` returns the current fill-settings in `FillInfo`

Errors: None.

See also: `SetFillPattern` ([544](#))

GetGraphMode

Synopsis: Get current graphical modus

Declaration: `function GetGraphMode : SmallInt`

Visibility: default

Description: `GetGraphMode` returns the current graphical modus

Errors: None.

See also: `InitGraph` ([539](#))

GetLineSettings

Synopsis: Get current line drawing settings

Declaration: `procedure GetLineSettings(var ActiveLineInfo: LineSettingsType)`

Visibility: default

Description: `GetLineSettings` returns the current Line settings in `LineInfo`

Errors: None.

See also: `SetLineStyle` ([545](#))

GetMaxColor

Synopsis: return maximum number of colors

Declaration: `function GetMaxColor : Word`

Visibility: default

Description: `GetMaxColor` returns the maximum color-number which can be set with `SetColor`. Contrary to Turbo Pascal, this color isn't always guaranteed to be white (for instance in 256+ color modes).

Errors: None.

See also: `SetColor` ([543](#)), `GetPaletteSize` ([536](#))

GetMaxMode

Synopsis: Return biggest mode for the current driver

Declaration: `function GetMaxMode : SmallInt`

Visibility: default

Description: `GetMaxMode` returns the highest modulus for the current driver.

Errors: None.

See also: `InitGraph` ([539](#))

GetMaxX

Synopsis: Return maximal X coordinate

Declaration: `function GetMaxX : SmallInt`

Visibility: default

Description: `GetMaxX` returns the maximum horizontal screen length

Errors: None.

See also: `GetMaxY` ([535](#))

GetMaxY

Synopsis: Return maximal Y coordinate

Declaration: `function GetMaxY : SmallInt`

Visibility: default

Description: `GetMaxY` returns the maximum number of screen lines

Errors: None.

See also: `GetMaxX` ([535](#))

GetModeName

Synopsis: Return description a modus

Declaration: `function GetModeName(ModeNumber: SmallInt) : String`

Visibility: default

Description: `GetModeName` Returns a string with the name of modus Modus

Errors: None.

See also: `GetDriverName` ([533](#)), `InitGraph` ([539](#))

GetModeRange

Synopsis: Return lowest and highest modus of current driver

Declaration: `procedure GetModeRange(GraphDriver: SmallInt; var LoMode: SmallInt;
var HiMode: SmallInt)`

Visibility: default

Description: `GetModeRange` returns the Lowest and Highest modus of the currently installed driver. If no modes are supported for this driver, `HiModus` will be -1.

Errors: None.

See also: `InitGraph` ([539](#))

GetPalette

Synopsis: Return current palette

Declaration: `procedure GetPalette(var Palette: PaletteType)`

Visibility: default

Description: `GetPalette` returns in `Palette` the current palette.

Errors: None.

See also: `GetPaletteSize` ([536](#)), `SetPalette` ([545](#))

GetPaletteSize

Synopsis: Return maximal number of entries in current palette

Declaration: `function GetPaletteSize : SmallInt`

Visibility: default

Description: `GetPaletteSize` returns the maximum number of entries in the current palette.

Errors: None.

See also: `GetPalette` ([536](#)), `SetPalette` ([545](#))

GetTextSettings

Synopsis: Return current text style

Declaration: `procedure GetTextSettings(var TextInfo: TextSettingsType)`

Visibility: default

Description: `GetTextSettings` returns the current text style settings : The font, direction, size and placement as set with `SetTextStyle` and `SetTextJustify`

Errors: None.

See also: `SetTextStyle` ([546](#)), `SetTextJustify` ([546](#))

GetViewSettings

Synopsis: Return current viewport

Declaration: `procedure GetViewSettings(var viewport: ViewPortType)`

Visibility: default

Description: `GetViewSettings` returns the current viewport and clipping settings in `ViewPort`.

Errors: None.

See also: `SetViewPort` ([547](#))

GetX

Synopsis: Return current cursor X position

Declaration: `function GetX : SmallInt`

Visibility: default

Description: `GetX` returns the X-coordinate of the current position of the graphical pointer

Errors: None.

See also: `GetY` ([537](#))

GetY

Synopsis: Return current cursor Y position

Declaration: `function GetY : SmallInt`

Visibility: default

Description: `GetY` returns the Y-coordinate of the current position of the graphical pointer

Errors: None.

See also: `GetX` ([537](#))

GraphDefaults

Synopsis: Reset graphical mode to defaults

Declaration: `procedure GraphDefaults`

Visibility: `default`

Description: `GraphDefaults` resets all settings for viewport, palette, foreground and background pattern, line-style and pattern, filling style, filling color and pattern, font, text-placement and text size.

Errors: None.

See also: `SetViewPort` ([547](#)), `SetFillStyle` ([544](#)), `SetColor` ([543](#)), `SetBkColor` ([543](#)), `SetLineStyle` ([545](#))

GraphErrorMsg

Synopsis: Return a description of an error

Declaration: `function GraphErrorMsg(ErrorCode: SmallInt) : String`

Visibility: `default`

Description: `GraphErrorMsg` returns a string describing the error `Errorcode`. This string can be used to let the user know what went wrong.

Errors: None.

See also: `GraphResult` ([538](#))

GraphResult

Synopsis: Result of last graphical operation

Declaration: `function GraphResult : SmallInt`

Visibility: `default`

Description: `GraphResult` returns an error-code for the last graphical operation. If the returned value is zero, all went well. A value different from zero means an error has occurred. besides all operations which draw something on the screen, the following procedures also can produce a `GraphResult` different from zero:

- `InstallUserFont` ([540](#))
- `SetLineStyle` ([545](#))
- `SetWriteMode` ([547](#))
- `SetFillStyle` ([544](#))
- `SetTextJustify` ([546](#))
- `SetGraphMode` ([545](#))
- `SetTextStyle` ([546](#))

Errors: None.

See also: `GraphErrorMsg` ([538](#))

InitGraph

Synopsis: Initialize graphical system

Declaration: `procedure InitGraph(var GraphDriver: SmallInt; var GraphMode: SmallInt;
const PathToDriver: String)`

Visibility: default

Description: `InitGraph` initializes the graph package. `GraphDriver` has two valid values: `GraphDriver=0` which performs an auto detect and initializes the highest possible mode with the most colors. 1024x768x64K is the highest possible resolution supported by the driver, if you need a higher resolution, you must edit `MODES.PPI`. If you need another mode, then set `GraphDriver` to a value different from zero and `graphmode` to the mode you wish (VESA modes where 640x480x256 is 101h etc.). `PathToDriver` is only needed, if you use the BGI fonts from Borland. Free Pascal does not offer BGI fonts like Borland, these must be obtained separately.

Example code:

```
var
  gd,gm : integer;
  PathToDriver : string;
begin
  gd:=detect; { highest possible resolution }
  gm:=0; { not needed, auto detection }
  PathToDriver:='C:\PP\BGI'; { path to BGI fonts,
                             drivers aren't needed }

  InitGraph(gd,gm,PathToDriver);
  if GraphResult<>grok then
    halt; ..... { whatever you need }
  CloseGraph; { restores the old graphics mode }
end.
```

Errors: None.

See also: Modes ([506](#)), DetectGraph ([530](#)), CloseGraph ([530](#)), GraphResult ([538](#))

InstallUserDriver

Synopsis: Install a user driver

Declaration: `function InstallUserDriver(Name: String; AutoDetectPtr: Pointer)
: SmallInt`

Visibility: default

Description: `InstallUserDriver` adds the device-driver `DriverPath` to the list of .BGI drivers. `AutoDetectPtr` is a pointer to a possible auto-detect function.

Errors: None.

See also: `InitGraph` ([539](#)), `InstallUserFont` ([540](#))

InstallUserFont

Synopsis: Install a user-defined font

Declaration: `function InstallUserFont(const FontFileName: String) : SmallInt`

Visibility: default

Description: `InstallUserFont` adds the font in `FontPath` to the list of fonts of the .BGI system.

Errors: None.

See also: `InitGraph` ([539](#)), `InstallUserDriver` ([539](#))

LineRel

Synopsis: Draw a line starting from current position in given direction

Declaration: `procedure LineRel(Dx: SmallInt; Dy: SmallInt)`

Visibility: default

Description: `LineRel` draws a line starting from the current pointer position to the point (DX, DY) , `\textbf{relative}` to the current position, in the current line style and color. The Current Position is set to the endpoint of the line.

Errors: None.

See also: `Line` ([528](#)), `LineTo` ([540](#))

LineTo

Synopsis: Draw a line starting from current position to a given point

Declaration: `procedure LineTo(X: SmallInt; Y: SmallInt)`

Visibility: default

Description: `LineTo` draws a line starting from the current pointer position to the point (DX, DY) , `\textbf{relative}` to the current position, in the current line style and color. The Current position is set to the end of the line.

Errors: None.

See also: `LineRel` ([540](#)), `Line` ([528](#))

MoveRel

Synopsis: Move cursor relative to current position

Declaration: `procedure MoveRel(Dx: SmallInt; Dy: SmallInt)`

Visibility: default

Description: `MoveRel` moves the pointer to the point (DX, DY) , relative to the current pointer position

Errors: None.

See also: `MoveTo` ([541](#))

MoveTo

Synopsis: Move cursor to absolute position.

Declaration: `procedure MoveTo(X: SmallInt;Y: SmallInt)`

Visibility: default

Description: `MoveTo` moves the pointer to the point (X,Y).

Errors: None.

See also: `MoveRel` ([540](#))

OutText

Synopsis: Write text on the screen at the current location.

Declaration: `procedure OutText(const TextString: String)`

Visibility: default

Description: `OutText` puts `TextString` on the screen, at the current pointer position, using the current font and text settings. The current position is moved to the end of the text.

Errors: None.

See also: `OutTextXY` ([528](#))

PieSlice

Synopsis: Draw a pie-slice

Declaration: `procedure PieSlice(X: SmallInt;Y: SmallInt;stangle: SmallInt;
 endAngle: SmallInt;Radius: Word)`

Visibility: default

Description: `PieSlice` draws and fills a sector of a circle with center (X,Y) and radius `Radius`, starting at angle `Start` and ending at angle `Stop`.

Errors: None.

See also: `Arc` ([529](#)), `Circle` ([527](#)), `Sector` ([543](#))

queryadapterinfo

Synopsis: Function called to retrieve the current video adapter settings.

Declaration: `function queryadapterinfo : PModeInfo`

Visibility: default

Rectangle

Synopsis: Draw a rectangle on the screen.

Declaration: `procedure Rectangle(x1: SmallInt; y1: SmallInt; x2: SmallInt; y2: SmallInt)`

Visibility: default

Description: Draws a rectangle with corners at (X1, Y1) and (X2, Y2), using the current color and style.

Errors: None.

See also: Bar ([529](#)), Bar3D ([530](#))

RegisterBGIDriver

Synopsis: Register a new BGI driver.

Declaration: `function RegisterBGIDriver(driver: pointer) : SmallInt`

Visibility: default

Description: Registers a user-defined BGI driver

Errors: None.

See also: InstallUserDriver ([539](#)), RegisterBGIFont ([542](#))

RegisterBGIfont

Synopsis: Register a new BGI font

Declaration: `function RegisterBGIfont(font: pointer) : SmallInt`

Visibility: default

Description: Registers a user-defined BGI driver

Errors: None.

See also: InstallUserFont ([540](#)), RegisterBGIDriver ([542](#))

RestoreCrtMode

Synopsis: Restore text screen

Declaration: `procedure RestoreCrtMode`

Visibility: default

Description: Restores the screen modus which was active before the graphical modus was started.

To get back to the graph mode you were last in, you can use `SetGraphMode(GetGraphMode)`

Errors: None.

See also: InitGraph ([539](#))

Sector

Synopsis: Draw and fill a sector of an ellipse

Declaration: `procedure Sector(x: SmallInt;y: SmallInt;StAngle: Word;EndAngle: Word;
 XRadius: Word;YRadius: Word)`

Visibility: default

Description: `Sector` draws and fills a sector of an ellipse with center (X, Y) and radii XRadius and YRadius, starting at angle Start and ending at angle Stop.

Errors: None.

See also: `Arc` ([529](#)), `Circle` ([527](#)), `PieSlice` ([541](#))

SetAspectRatio

Synopsis: Set aspect ration of the screen

Declaration: `procedure SetAspectRatio(Xasp: Word;Yasp: Word)`

Visibility: default

Description: Sets the aspect ratio of the current screen to Xasp/Yasp.

Errors: None

See also: `InitGraph` ([539](#)), `GetAspectRatio` ([532](#))

SetBkColor

Synopsis: Set background drawing color

Declaration: `procedure SetBkColor(ColorNum: Word)`

Visibility: default

Description: Sets the background color to Color.

Errors: None.

See also: `GetBkColor` ([532](#)), `SetColor` ([543](#)), `SetWriteMode` ([547](#))

SetColor

Synopsis: Set foreground drawing color

Declaration: `procedure SetColor(Color: Word)`

Visibility: default

Description: Sets the foreground color to Color.

Errors: None.

See also: `GetColor` ([533](#)), `SetBkColor` ([543](#)), `SetWriteMode` ([547](#))

SetDirectVideo

Synopsis: Attempt to enter direct video mode.

Declaration: `procedure SetDirectVideo(DirectAccess: Boolean)`

Visibility: default

Description: `SetDirectVideo` attempts to enter direct video mode. In that mode, everything is drawn straight in the video buffer.

SetFillPattern

Synopsis: Set drawing fill pattern

Declaration: `procedure SetFillPattern(Pattern: FillPatternType; Color: Word)`

Visibility: default

Description: `SetFillPattern` sets the current fill-pattern to `FillPattern`, and the filling color to `Color`. The pattern is an 8x8 raster, corresponding to the 64 bits in `FillPattern`.

Errors: None

See also: `GetFillPattern` ([534](#)), `SetFillStyle` ([544](#)), `SetWriteMode` ([547](#))

SetFillStyle

Synopsis: Set drawing fill style

Declaration: `procedure SetFillStyle(Pattern: Word; Color: Word)`

Visibility: default

Description: `SetFillStyle` sets the filling pattern and color to one of the predefined filling patterns. `Pattern` can be one of the following predefined constants :

EmptyFill Uses backgroundcolor.

SolidFill Uses filling color

LineFill Fills with horizontal lines.

LtSlashFill Fills with lines from left-under to top-right.

SlashFill Idem as previous, thick lines.

BkSlashFill Fills with thick lines from left-Top to bottom-right.

LtBkSlashFill Idem as previous, normal lines.

HatchFill Fills with a hatch-like pattern.

XHatchFill Fills with a hatch pattern, rotated 45 degrees.

InterLeaveFill

WideDotFill Fills with dots, wide spacing.

CloseDotFill Fills with dots, narrow spacing.

UserFill Fills with a user-defined pattern.

Errors: None.

See also: `SetFillPattern` ([544](#)), `SetWriteMode` ([547](#))

SetGraphMode

Synopsis: Set graphical mode

Declaration: `procedure SetGraphMode(Mode: SmallInt)`

Visibility: default

Description: `SetGraphMode` sets the graphical mode and clears the screen.

Errors: None.

See also: `InitGraph` ([539](#))

SetLineStyle

Synopsis: Set line drawing style

Declaration: `procedure SetLineStyle(LineStyle: Word;Pattern: Word;Thickness: Word)`

Visibility: default

Description: `SetLineStyle` sets the drawing style for lines. You can specify a `LineStyle` which is one of the following pre-defined constants:

Solidln Draws a solid line.

Dottedln Draws a dotted line.

Centerln Draws a non-broken centered line.

Dashedln Draws a dashed line.

UserBitln Draws a User-defined bit pattern.

If `UserBitln` is specified then `Pattern` contains the bit pattern. In all another cases, `Pattern` is ignored. The parameter `Width` indicates how thick the line should be. You can specify one of the following pre-defined constants:

NormWidth Normal line width

ThickWidth Double line width

Errors: None.

See also: `GetLineSettings` ([534](#)), `SetWriteMode` ([547](#))

SetPalette

Synopsis: Set palette entry using color constant

Declaration: `procedure SetPalette(ColorNum: Word;Color: ShortInt)`

Visibility: default

Description: `SetPalette` changes the `ColorNr`-th entry in the palette to `NewColor`

Errors: None.

See also: `SetAllPalette` ([529](#)), `SetRGBPalette` ([529](#))

SetTextJustify

Synopsis: Set text placement style

Declaration: `procedure SetTextJustify(horiz: Word;vert: Word)`

Visibility: default

Description: `SetTextJustify` controls the placement of new text, relative to the (graphical) cursor position. `Horizontal` controls horizontal placement, and can be one of the following pre-defined constants:

LeftTextText is set left of the pointer.

CenterTextText is set centered horizontally on the pointer.

RightTextText is set to the right of the pointer.

`Vertical` controls the vertical placement of the text, relative to the (graphical) cursor position. Its value can be one of the following pre-defined constants :

BottomTextText is placed under the pointer.

CenterTextText is placed centered vertically on the pointer.

TopTextText is placed above the pointer.

Errors: None.

See also: `OutText` ([541](#)), `OutTextXY` ([528](#))

SetTextStyle

Synopsis: Set text style

Declaration: `procedure SetTextStyle(font: Word;direction: Word;charsize: Word)`

Visibility: default

Description: `SetTextStyle` controls the style of text to be put on the screen. pre-defined constants for `Font` are:

DefaultFontThe default font

TriplexFontA special font

SmallFontA smaller font

SansSerifFontA sans-serif font (like Arial)

GothicFontA gothic font

ScriptFontA script font

SimpleFontA simple font

TSCRFontTerminal screen font

LCOMFont?

EuroFont?

BoldFontA bold typeface font

Pre-defined constants for `Direction` are :

HorizDirWrite horizontal

VertDirWrite vertical

Errors: None.

See also: `GetTextSettings` ([537](#))

SetUserCharSize

Synopsis: Set user character size for vector font

Declaration: `procedure SetUserCharSize(Multx: Word; Divx: Word; Multy: Word; Divy: Word)`

Visibility: default

Description: Sets the width and height of vector-fonts. The horizontal size is given by $Xasp1/Xasp2$, and the vertical size by $Yasp1/Yasp2$.

Errors: None.

See also: [SetTextStyle \(546\)](#)

SetViewPort

Synopsis: Set the graphical drawing window

Declaration: `procedure SetViewPort(X1: SmallInt; Y1: SmallInt; X2: SmallInt;
Y2: SmallInt; Clip: Boolean)`

Visibility: default

Description: Sets the current graphical viewport (window) to the rectangle defined by the top-left corner (X1, Y1) and the bottom-right corner (X2, Y2). If Clip is true, anything drawn outside the viewport (window) will be clipped (i.e. not drawn). Coordinates specified after this call are relative to the top-left corner of the viewport.

Errors: None.

See also: [GetViewSettings \(537\)](#)

SetWriteMode

Synopsis: Specify binary operation to perform when drawing on screen

Declaration: `procedure SetWriteMode(WriteMode: SmallInt)`

Visibility: default

Description: `SetWriteMode` controls the drawing of lines on the screen. It controls the binary operation used when drawing lines on the screen. Mode can be one of the following pre-defined constants:

CopyPut Draw as specified using current bitmask and color

XORPut Draw XOR-ing current bitmask and color

Errors: None.

See also: [SetColor \(543\)](#), [SetBkColor \(543\)](#), [SetLineStyle \(545\)](#), [SetFillStyle \(544\)](#)

TextHeight

Synopsis: Return height (in pixels) of the given string

Declaration: `function TextHeight(const TextString: String) : Word`

Visibility: default

Description: `TextHeight` returns the height (in pixels) of the string `S` in the current font and text-size.

Errors: None.

See also: `TextWidth` ([548](#))

TextWidth

Synopsis: Return width (in pixels) of the given string

Declaration: `function TextWidth(const TextString: String) : Word`

Visibility: default

Description: `TextWidth` returns the width (in pixels) of the string `S` in the current font and text-size.

Errors: None.

See also: `TextHeight` ([548](#))

Chapter 13

Reference for unit 'heaptrc'

13.1 Controlling HeapTrc with environment variables

The `HeapTrc` unit can be controlled with the `HEAPTRC` environment variable. The contents of this variable controls the initial setting of some constants in the unit. `HEAPTRC` consists of one or more of the following strings, separated by spaces:

keepreleased If this string occurs, then the `KeepReleased` (551) variable is set to `True`

disabled If this string occurs, then the `UseHeapTrace` (551) variable is set to `False` and the heap trace is disabled. It does not make sense to combine this value with other values.

nohalt If this string occurs, then the `HaltOnError` (550) variable is set to `False`, so the program continues executing even in case of a heap error.

log=filename If this string occurs, then the output of `heaptrc` is sent to the specified `Filename`. (see also `SetHeapTraceOutput` (553))

The following are valid values for the `HEAPTRC` variable:

```
HEAPTRC=disabled
HEAPTRC="keepreleased log=heap.log"
HEAPTRC="log=myheap.log nohalt"
```

Note that these strings are case sensitive, and the name of the variable too.

13.2 HeapTrc Usage

All that you need to do is to include `heaptrc` in the `uses` clause of your program. Make sure that it is the first unit in the clause, otherwise memory allocated in initialization code of units that precede the `heaptrc` unit will not be accounted for, causing an incorrect memory usage report.

If you use the `-gh` switch, the compiler will insert the unit by itself, so you don't have to include it in your `uses` clause.

The below example shows how to use the `heaptrc` unit.

This is the memory dump shown when running this program in a standard way:

```

Marked memory at 0040FA50 invalid
Wrong size : 128 allocated 64 freed
  0x00408708
  0x0040CB49
  0x0040C481
Call trace for block 0x0040FA50 size 128
  0x0040CB3D
  0x0040C481

```

If you use the `lineinfo` unit (or use the `-gl` switch) as well, then `heaptrc` will also give you the filenames and line-numbers of the procedures in the backtrace:

```

Marked memory at 00410DA0 invalid
Wrong size : 128 allocated 64 freed
  0x004094B8
  0x0040D8F9  main,   line 25 of heapex.pp
  0x0040D231
Call trace for block 0x00410DA0 size 128
  0x0040D8ED  main,   line 23 of heapex.pp
  0x0040D231

```

If lines without filename/line-number occur, this means there is a unit which has no debug info included.

13.3 Overview

This document describes the HEAPTRC unit for Free Pascal. It was written by Pierre Muller. It is system independent, and works on all supported systems.

The HEAPTRC unit can be used to debug your memory allocation/deallocation. It keeps track of the calls to `getmem/freemem`, and, implicitly, of `New/Dispose` statements.

When the program exits, or when you request it explicitly. It displays the total memory used, and then dumps a list of blocks that were allocated but not freed. It also displays where the memory was allocated.

If there are any inconsistencies, such as memory blocks being allocated or freed twice, or a memory block that is released but with wrong size, this will be displayed also.

The information that is stored/displayed can be customized using some constants.

13.4 Constants, types and variables

Constants

```
add_tail : Boolean = true
```

If `add_tail` is `True` (the default) then a check is also performed on the memory location just behind the allocated memory.

```
HaltOnError : Boolean = true
```

If `HaltOnError` is set to `True` then an illegal call to `FreeMem` will cause the memory manager to execute a `halt(1)` instruction, causing a memory dump. By Default it is set to `True`.

```
keepreleased : Boolean = false
```

If `keepreleased` is set to true, then a list of freed memory blocks is kept. This is useful if you suspect that the same memory block is released twice. However, this option is very memory intensive, so use it sparingly, and only when it's really necessary.

```
quicktrace : Boolean = true
```

`Quicktrace` determines whether the memory manager checks whether a block that is about to be released is allocated correctly. This is a rather time consuming search, and slows program execution significantly, so by default it is set to `True`.

```
tracesize = 8
```

`Tracesize` specifies how many levels of calls are displayed of the call stack during the memory dump. If you specify `keepreleased:=True` then half the `TraceSize` is reserved for the `GetMem` call stack, and the other half is reserved for the `FreeMem` call stack. For example, the default value of 8 will cause eight levels of call frames to be dumped for the `getmem` call if `keepreleased` is `False`. If `KeepReleased` is true, then 4 levels of call frames will be dumped for the `GetMem` call and 4 frames will be dumped for the `FreeMem` call. If you want to change this value, you must recode the `heaptrc` unit.

```
usecrc : Boolean = true
```

If `usecrc` is `True` (the default) then a crc check is performed on locations before and after the allocated memory. This is useful to detect memory overwrites.

```
useheaptrace : Boolean = true
```

This variable must be set at program startup, through the help of an environment variable.

Types

```
tDisplayExtraInfoProc = procedure(var ptext: text;p: pointer)
```

The `TDisplayExtraInfoType` is a procedural type used in the `SetHeapExtraInfo` (552) call to display a memory location which was previously filled with `TFillExtraInfoProc` (551)

```
tFillExtraInfoProc = procedure(p: pointer)
```

The `TFillExtraInfoProc` is a procedural type used in the `SetHeapExtraInfo` (552) call to fill a memory location with extra data for displaying.

13.5 Procedures and functions

DumpHeap

Synopsis: Dump memory usage report to stderr.

Declaration: `procedure DumpHeap`

Visibility: default

Description: DumpHeap dumps to standard output a summary of memory usage. It is called automatically by the heaptrc unit when your program exits (by installing an exit procedure), but it can be called at any time.

Errors: None.

See also: MarkHeap ([552](#))

MarkHeap

Synopsis: Mark memory blocks with a signature.

Declaration: `procedure MarkHeap`

Visibility: default

Description: MarkHeap marks all memory blocks with a special signature. You can use this if you think that you corrupted the memory.

Errors: None.

See also: DumpHeap ([551](#))

SetHeapExtraInfo

Synopsis: Store extra information in blocks.

Declaration: `procedure SetHeapExtraInfo(size: ptrint; fillproc: tFillExtraInfoProc;
displayproc: tdisplayextrainfoProc)`

Visibility: default

Description: You can use SetHeapExtraInfo to store extra info in the blocks that the heaptrc unit reserves when tracing getmem calls. Size indicates the size (in bytes) that the trace mechanism should reserve for your extra information. For each call to getmem, FillProc will be called, and passed a pointer to the memory reserved.

When dumping the memory summary, the extra info is shown by calling displayproc and passing it the memory location which was filled by fillproc. It should write the information in readable form to the text file provided in the call to displayproc

Errors: You can only call SetHeapExtraInfo if no memroy has been allocated yet. If memory was already allocated prior to the call to SetHeapExtraInfo, then an error will be displayed on standard error output, and a DumpHeap ([551](#)) is executed.

See also: DumpHeap ([551](#)), SetHeapTraceOutput ([553](#))

Listing: `./heapex/setinfo.pp`

Program heapex;

{ Program used to demonstrate the usage of heaptrc unit }

Uses heaptrc;

Var P1 : ^Longint;
P2 : Pointer;
I : longint;
Marker : Longint;

```

Procedure SetMarker (P : pointer);

Type PLongint = ^Longint;

begin
    PLongint(P)^:=Marker;
end;

Procedure Part1;

begin
    // Blocks allocated here are marked with $FFAAFFAA = -5570646
    Marker := $FFAAFFAA;
    New(P1);
    New(P1);
    Dispose(P1);
    For I:=1 to 10 do
        begin
            GetMem (P2,128);
            If (I mod 2) = 0 Then FreeMem(P2,128);
            end;
        GetMem(P2,128);
    end;

Procedure Part2;

begin
    // Blocks allocated here are marked with $FAFAFAFA = -84215046
    Marker := $FAFAFAFA;
    New(P1);
    New(P1);
    Dispose(P1);
    For I:=1 to 10 do
        begin
            GetMem (P2,128);
            If (I mod 2) = 0 Then FreeMem(P2,128);
            end;
        GetMem(P2,128);
    end;

begin
    SetExtraInfo (SizeOf(Marker) , @SetMarker);
    Writeln ( 'Part 1 ');
    part1;
    Writeln ( 'Part 2 ');
    part2;
end.

```

SetHeapTraceOutput

Synopsis: Specify filename for heap trace output.

Declaration: `procedure SetHeapTraceOutput(const name: String)`

Visibility: default

Description: `SetHeapTraceOutput` sets the filename into which heap trace info will be written. By default information is written to standard output, this function allows you to redirect the information to a file with full filename name.

Errors: If the file cannot be written to, errors will occur when writing the trace.

See also: `SetHeapExtraInfo` ([552](#))

Chapter 14

Reference for unit 'ipc'

14.1 Used units

Table 14.1: Used units by unit 'ipc'

Name	Page
BaseUnix	555

14.2 Overview

This document describes the IPC unit for Free Pascal. It was written for linux by Michael Van Canneyt. It gives all the functionality of system V Inter-Process Communication: shared memory, semaphores and messages. It works only on the linux operating system.

Many constants here are provided for completeness only, and should under normal circumstances not be used by the programmer.

14.3 Constants, types and variables

Constants

`IPC_CREAT = 1 shl 9`

Create if key is nonexistent

`IPC_EXCL = 2 shl 9`

fail if key exists

`IPC_INFO = 3`

For ipcs call

`IPC_NOWAIT = 4 shl 9`

return error on wait

IPC_RMID = 0

Remove resource

IPC_SET = 1

set ipc_perm options

IPC_STAT = 2

get ipc_perm options

MSGMAX = 4056

Internal Message control code. Do not use

MSGMNB = 16384

Internal Message control code. Do not use

MSGMNI = 128

Internal Message control code. Do not use

MSG_EXCEPT = 2 shl 12

Internal Message control code. Do not use

MSG_NOERROR = 1 shl 12

Internal Message control code. Do not use

SEM_GETALL = 13

Semaphore operation: Get all semaphore values

SEM_GETNCNT = 14

Semaphore operation: Get number of processes waiting for resource.

SEM_GETPID = 11

Semaphore operation: Get process ID of last operation.

SEM_GETVAL = 12

Semaphore operation: Get current value of semaphore

SEM_GETZCNT = 15

Semaphore operation: Get number of processes waiting for semaphores to reach zero

SEM_SEMNI = 128

Semaphore operation: ?

`SEM_SEMNS = (SEM_SEMNI * SEM_SEMMSL)`

Semaphore operation: ?

`SEM_SEMMSL = 32`

Semaphore operation: ?

`SEM_SEMOPM = 32`

Semaphore operation: ?

`SEM_SEMVMX = 32767`

Semaphore operation: ?

`SEM_SETALL = 17`

Semaphore operation: Set all semaphore values

`SEM_SETVAL = 16`

Semaphore operation: Set semaphore value

`SEM_UNDO = $1000`

Constant for use in `semop` ([571](#))

`SHM_LOCK = 11`

This constant is used in the `shmctl` ([572](#)) call.

`SHM_R = 4 shl 6`

This constant is used in the `shmctl` ([572](#)) call.

`SHM_RDONLY = 1 shl 12`

This constant is used in the `shmctl` ([572](#)) call.

`SHM_REMAP = 4 shl 12`

This constant is used in the `shmctl` ([572](#)) call.

`SHM_RND = 2 shl 12`

This constant is used in the `shmctl` ([572](#)) call.

`SHM_UNLOCK = 12`

This constant is used in the `shmctl` ([572](#)) call.

`SHM_W = 2 shl 6`

This constant is used in the `shmctl` ([572](#)) call.

Types

`key_t` = `TKey`

Alias for `TKey` (559) type

`msglen_t` = `culong`

Message length type

`msgqnum_t` = `culong`

Message queue number type

`PIPC_Perm` = `^TIPC_Perm`

Pointer to `TIPC_Perm` (559) record.

`PMSG` = `^TMSG`

Pointer to `TMSG` (559) record

`PMSGbuf` = `^TMSGbuf`

Pointer to `TMsgBuf` (559) record

`PMSGinfo` = `^TMSGinfo`

Pointer to `TMSGinfo` (559) record

`PMSQid_ds` = `^TMSQid_ds`

Pointer to `TMSQid_ds` (560)

`PSEMBuf` = `^TSEMBuf`

Pointer to `TSembuf` (560) record.

`PSEMid_ds` = `^TSEMid_ds`

Pointer to `TSEMid_ds` (560) record.

`PSEMinfo` = `^TSEMinfo`

Pointer to `TSEMinfo` (561) record.

`PSEMun` = `^TSEMun`

Pointer to `TSEMun` (561) record

`PShmid_DS` = `^TShmid_ds`

Pointer to `TSHMid_ds` (561) record.

`PSHMinfo` = `^TSHMinfo`

```
TIPC_Perm = record
  key : TKey;
  uid : Word;
  gid : Word;
  cuid : Word;
  cgid : Word;
  mode : Word;
  seq : Word;
end
```

TIPC_Perm is used in all IPC systems to specify the permissions. It should never be used directly.

```
TKey = LongInt
```

Type returned by the ftok (561) key generating function.

```
TMSG = record
  msg_next : PMSG;
  msg_type : LongInt;
  msg_spot : PChar;
  msg_stime : LongInt;
  msg_ts : Integer;
end
```

Record used in the handling of message queues. Do not use directly.

```
TMSGbuf = record
  mtype : LongInt;
  mtext : Array[0..0] of Char;
end
```

The TMSGbuf record is a record containing the data of a record. you should never use this record directly, instead you should make your own record that follows the structure of the TMSGbuf record, but that has a size that is big enough to accomodate your messages. The mtype field should always be present, and should always be filled.

```
TMSGinfo = record
  msgpool : LongInt;
  msgmap : LongInt;
  msgmax : LongInt;
  msgmnb : LongInt;
  msgmni : LongInt;
  msgssz : LongInt;
  msgtql : LongInt;
  msgseg : Word;
end
```

Internal message system record. Do not use directly.

```
TMSQid_ds = record
```

```

msg_perm : TIPC_Perm;
msg_first : PMSG;
msg_last : PMSG;
msg_stime : LongInt;
msg_rtime : LongInt;
msg_ctime : LongInt;
wwait : Pointer;
rwait : pointer;
msg_cbytes : Word;
msg_qnum : Word;
msg_qbytes : Word;
msg_lspid : Word;
msg_lrpid : Word;
end

```

This record should never be used directly, it is an internal kernel record. It's fields may change at any time.

```

TSEMbuf = record
  sem_num : cushort;
  sem_op : cshort;
  sem_flg : cshort;
end

```

The TSEMbuf record is used in the semop ([571](#)) call, and is used to specify which operations you want to do.

```

TSEMid_ds = record
  sem_perm : TIPC_Perm;
  sem_otime : LongInt;
  sem_ctime : LongInt;
  sem_base : pointer;
  sem_pending : pointer;
  sem_pending_last : pointer;
  undo : pointer;
  sem_nsems : Word;
end

```

Structure returned by the semctl ([566](#)) call, contains all data of a semaphore

```

TSEMinfo = record
  semmap : cint;
  semmni : cint;
  semmns : cint;
  semmnu : cint;
  semmsl : cint;
  semopm : cint;
  semume : cint;
  semusz : cint;
  semvmx : cint;
  semaem : cint;
end

```

end

Internal semaphore system record. Do not use.

```
TSEMun = record
end
```

Record used in semctl (566) call.

```
TShmid_ds = record
  shm_perm : TIPC_Perm;
  shm_segsz : LongInt;
  shm_atime : LongInt;
  shm_dtime : LongInt;
  shm_ctime : LongInt;
  shm_cpid : Word;
  shm_lpid : Word;
  shm_nattch : Integer;
  shm_npages : Word;
  shm_pages : Pointer;
  attaches : pointer;
end
```

Record used in the shmctl (572) call to set or retrieve settings for shared memory.

```
TSHMinfo = record
  shmmax : cint;
  shmmmin : cint;
  shmmni : cint;
  shmseg : cint;
  shmall : cint;
end
```

Record used by the shared memory system, Do not use directly.

14.4 Procedures and functions

ftok

Synopsis: Create token from filename

Declaration: function ftok(Path: pchar; ID: cint) : TKey

Visibility: default

Description: ftok returns a key that can be used in a semget (570)shmget (574) or msgget (564) call to access a new or existing IPC resource.

Path is the name of a file in the file system, ID is a character of your choice. The ftok call does the same as it's C counterpart, so a pascal program and a C program will access the same resource if they use the same Path and ID

For an example, see msgctl (562), semctl (566) or shmctl (572).

Errors: `ftok` returns -1 if the file in `Path` doesn't exist.

See also: `semget` (570), `shmget` (574), `msgget` (564)

msgctl

Synopsis: Perform various operations on a message queue

Declaration: `function msgctl(msgid: cint;cmd: cint;buf: PMSQid_ds) : cint`

Visibility: default

Description: `msgctl` performs various operations on the message queue with id `ID`. Which operation is performed, depends on the `cmd` parameter, which can have one of the following values:

IPC_STATIn this case, the `msgctl` call fills the `TMSQid_ds` structure with information about the message queue.

IPC_SETIn this case, the `msgctl` call sets the permissions of the queue as specified in the `ipc_perm` record inside `buf`.

IPC_RMIDIf this is specified, the message queue will be removed from the system.

`buf` contains the data that are needed by the call. It can be `Nil` in case the message queue should be removed.

The function returns `True` if successfull, `False` otherwise.

Errors: On error, `False` is returned, and `IPCError` is set accordingly.

See also: `msgget` (564), `msgsnd` (565), `msgrcv` (565)

Listing: `./ipcex/msgtool.pp`

```

program msgtool;

Uses ipc;

Type
  PMyMsgBuf = ^TMyMsgBuf;
  TMyMsgBuf = record
    mtype : Longint;
    mtext : string[255];
  end;

Procedure DoError (Const Msg : string);

begin
  WriteLn (msg, 'returned an error : ',ipcerror);
  halt(1);
end;

Procedure SendMessage (Id : Longint;
  Var Buf : TMyMsgBuf;
  MType : Longint;
  Const MText : String);

begin
  WriteLn ( 'Sending message. ');
  Buf.mtype:=mtype;

```

```

    Buf.Mtext:=mtext;
    If not msgsnd(Id,PMsgBuf(@Buf),256,0) then
        DoError('msgsnd');
end;

Procedure ReadMessage (ID : Longint;
                        Var Buf : TMyMsgBuf;
                        MType : longint);

begin
    Writeln ('Reading message. ');
    Buf.MType:=MType;
    If msgrcv(ID,PMSGBuf(@Buf),256,mtype,0) then
        Writeln ('Type : ',buf.mtype,' Text : ',buf.mtext)
    else
        DoError ('msgrcv');
end;

Procedure RemoveQueue ( ID : Longint);

begin
    If msgctl (id,IPC_RMID,Nil) then
        Writeln ('Removed Queue with id ',Id);
end;

Procedure ChangeQueueMode (ID,mode : longint);

Var QueueDS : TMSQid_ds;

begin
    If Not msgctl (Id,IPC_STAT,@QueueDS) then
        DoError ('msgctl : stat');
    Writeln ('Old permissions : ',QueueDS.msg_perm.mode);
    QueueDS.msg_perm.mode:=Mode;
    if msgctl (ID,IPC_SET,@QueueDS) then
        Writeln ('New permissions : ',QueueDS.msg_perm.mode)
    else
        DoError ('msgctl : IPC_SET');
end;

procedure usage;

begin
    Writeln ('Usage : msgtool s(end) <type> <text> (max 255 characters)');
    Writeln ('                      r(eceive) <type>');
    Writeln ('                      d(elete)');
    Writeln ('                      m(ode) <decimal mode>');
    halt(1);
end;

Function StrToInt (S : String): longint;

Var M : longint;
    C : Integer;

begin
    val (S,M,C);
    If C<>0 Then DoError ('StrToInt : '+S);

```

```

StrToInt:=M;
end;

Var
  Key : TKey;
  ID : longint;
  Buf : TMyMsgBuf;

begin
  If Paramcount<1 then Usage;
  key := Ftok( ' ', 'M' );
  ID:=msgget(key,IPC_CREAT or 438);
  If ID<0 then DoError ( 'MsgGet' );
  Case upCase(Paramstr(1)[1]) of
    'S' : If ParamCount<>3 then
      Usage
    else
      SendMessage ( id , Buf , StrToInt (Paramstr(2)) , paramstr(3));
    'R' : If ParamCount<>2 then
      Usage
    else
      ReadMessage ( id , buf , strtoint (Paramstr(2)));
    'D' : If ParamCount<>1 then
      Usage
    else
      RemoveQueue ( ID );
    'M' : If ParamCount<>2 then
      Usage
    else
      ChangeQueueMode ( id , strtoint (paramstr(2)));
  else
    Usage
  end;
end.

```

msgget

Synopsis: Return message queue ID, possibly creating the queue

Declaration: function msgget(key: TKey;msgflg: cint) : cint

Visibility: default

Description: msgget returns the ID of the message queue described by key. Depending on the flags in msgflg, a new queue is created.

msgflg can have one or more of the following values (combined by ORs):

IPC_CREATThe queue is created if it doesn't already exist.

IPC_EXCLIf used in combination with IPC_CREAT, causes the call to fail if the queue already exists. It cannot be used by itself.

Optionally, the flags can be ORed with a permission mode, which is the same mode that can be used in the file system.

For an example, see msgctl ([562](#)).

Errors: On error, -1 is returned, and IPCError is set.

See also: ftok ([561](#)), msgsnd ([565](#)), msgrcv ([565](#)), msgctl ([562](#))

msgrcv

Synopsis: Retrieve a message from the queue

Declaration: `function msgrcv(msqid: cint;msgp: PMSGbuf;msgsz: size_t;msgtyp: cint;
msgflg: cint) : cint`

Visibility: default

Description: `msgrcv` retrieves a message of type `msgtyp` from the message queue with ID `msqid`. `msgtyp` corresponds to the `mtype` field of the `TMSGbuf` record. The message is stored in the `MSGbuf` structure pointed to by `msgp`.

The `msgflg` parameter can be used to control the behaviour of the `msgrcv` call. It consists of an ORed combination of the following flags:

0No special meaning.

IPC_NOWAITIf no messages are available, then the call returns immediately, with the `ENOMSG` error.

MSG_NOERRORIf the message size is wrong (too large), no error is generated, instead the message is truncated. Normally, in such cases, the call returns an error (`E2BIG`)

The function returns `True` if the message was received correctly, `False` otherwise.

For an example, see `msgctl` ([562](#)).

Errors: In case of error, `False` is returned, and `IPCError` is set.

See also: `msgget` ([564](#)), `msgsnd` ([565](#)), `msgctl` ([562](#))

msgsnd

Synopsis: Send a message to the message queue

Declaration: `function msgsnd(msqid: cint;msgp: PMSGbuf;msgsz: size_t;msgflg: cint)
: cint`

Visibility: default

Description: `msgsend` sends a message to a message queue with ID `msqid`. `msgp` is a pointer to a message buffer, that should be based on the `TMsgBuf` type. `msgsz` is the size of the message (NOT of the message buffer record !)

The `msgflg` can have a combination of the following values (ORed together):

0No special meaning. The message will be written to the queue. If the queue is full, then the process is blocked.

IPC_NOWAITIf the queue is full, then no message is written, and the call returns immediately.

The function returns `True` if the message was sent successfully, `False` otherwise.

For an example, see `msgctl` ([562](#)).

Errors: In case of error, the call returns `False`, and `IPCError` is set.

See also: `msgget` ([564](#)), `msgrcv` ([565](#)), `msgctl` ([562](#))

semctl

Synopsis: Perform various control operations on a semaphore set

Declaration: `function semctl(semid: cint; semnum: cint; cmd: cint; var arg: TSEMun) : LongInt`

Visibility: default

Description: `semctl` performs various operations on the semaphore `semnum` with semaphore set id `ID`.

The `arg` parameter supplies the data needed for each call. This is a variant record that should be filled differently, according to the command:

```
Type
TSEMun = record
  case longint of
    0 : ( val : longint );
    1 : ( buf : PSEMid_ds );
    2 : ( arr : PWord );
    3 : ( padbuf : PSeminfo );
    4 : ( padpad : pointer );
  end;
```

Which operation is performed, depends on the `cmd` parameter, which can have one of the following values:

IPC_STATIn this case, the `arg` record should have its `buf` field set to the address of a `TSEMid_ds` record. The `semctl` call fills this `TSEMid_ds` structure with information about the semaphore set.

IPC_SETIn this case, the `arg` record should have its `buf` field set to the address of a `TSEMid_ds` record. The `semctl` call sets the permissions of the queue as specified in the `ipc_perm` record.

IPC_RMIDIf this is specified, the semaphore set is removed from the system.

GETALLIn this case, the `arr` field of `arg` should point to a memory area where the values of the semaphores will be stored. The size of this memory area is `\var{SizeOf(Word)* Number of semaphores in the set}`. This call will then fill the memory array with all the values of the semaphores.

GETNCNTThis will fill the `val` field of the `arg` union with the number of processes waiting for resources.

GETPID`semctl` returns the process ID of the process that performed the last `semop` (571) call.

GETVAL`semctl` returns the value of the semaphore with number `semnum`.

GETZCNT`semctl` returns the number of processes waiting for semaphores that reach value zero.

SETALLIn this case, the `arr` field of `arg` should point to a memory area where the values of the semaphores will be retrieved from. The size of this memory area is `\var{SizeOf(Word)* Number of semaphores in the set}`. This call will then set the values of the semaphores from the memory array.

SETVALThis will set the value of semaphore `semnum` to the value in the `val` field of the `arg` parameter.

The function returns -1 on error.

Errors: The function returns -1 on error, and `IPCError` is set accordingly.

See also: [semget \(570\)](#), [semop \(571\)](#)

Listing: ./ipcex/semtool.pp

```

Program semtool;

{ Program to demonstrat the use of semaphores }

Uses ipc;

Const MaxSemValue = 5;

Procedure DoError (Const Msg : String);

begin
  WriteLn ( 'Error : ',msg, ' Code : ',IPCError);
  Halt(1);
end;

Function getsemval (ID,Member : longint) : longint;

Var S : TSEMun;

begin
  GetSemVal:=SemCtl(id ,member,GETVAL,S);
end;

Procedure DispVal (ID,member : longint);

begin
  writeln ( 'Value for member ',member, ' is ',GetSemVal(ID,Member));
end;

Function GetMemberCount (ID : Longint) : longint;

Var opts : TSEMun;
    semds : TSEMid_ds;

begin
  opts.buf:=@semds;
  If semctl(Id,0,IPC_STAT,opts)<>-1 then
    GetMemberCount:=semds.sem_nsems
  else
    GetMemberCount:=-1;
end;

Function OpenSem (Key : TKey) : Longint;

begin
  OpenSem:=semget(Key,0,438);
  If OpenSem=-1 then
    DoError ( 'OpenSem' );
end;

Function CreateSem (Key : TKey; Members : Longint) : Longint;

Var Count : Longint;
    Semopts : TSemun;

```

```

begin
  If members>semmsl then
    DoError ( 'Sorry , maximum number of semaphores in set exceeded');
  WriteIn ( 'Trying to create a new semaphore set with ',members,' members. ');
  CreateSem:=semget(key,members,IPC_CREAT or IPC_Excl or 438);
  If CreateSem=-1 then
    DoError ( 'Semaphore set already exists. ');
  Semopts.val:=MaxSemValue; { Initial value of semaphores }
  For Count:=0 to Members-1 do
    semctl(CreateSem,count,setval,semopts);
end;

```

Procedure lockSem (ID,Member: Longint);

Var lock : TSEMbuf;

```

begin
  With lock do
    begin
      sem_num:=0;
      sem_op:=-1;
      sem_flg:=IPC_NOWAIT;
    end;
    if (member<0) or (member>GetMemberCount(ID)-1) then
      DoError ( 'semaphore member out of range');
    if getsemval(ID,member)=0 then
      DoError ( 'Semaphore resources exhausted (no lock)');
    lock.sem_num:=member;
    WriteIn ( 'Attempting to lock member ',member, ' of semaphore ',ID);
    if not semop(Id,@lock,1) then
      DoError ( 'Lock failed')
    else
      WriteIn ( 'Semaphore resources decremented by one');
      dispval(ID,Member);
end;

```

Procedure UnlockSem (ID,Member: Longint);

Var Unlock : TSEMbuf;

```

begin
  With Unlock do
    begin
      sem_num:=0;
      sem_op:=1;
      sem_flg:=IPC_NOWAIT;
    end;
    if (member<0) or (member>GetMemberCount(ID)-1) then
      DoError ( 'semaphore member out of range');
    if getsemval(ID,member)=MaxSemValue then
      DoError ( 'Semaphore not locked');
    Unlock.sem_num:=member;
    WriteIn ( 'Attempting to unlock member ',member, ' of semaphore ',ID);
    if not semop(Id,@unlock,1) then
      DoError ( 'Unlock failed')
    else
      WriteIn ( 'Semaphore resources incremented by one');
      dispval(ID,Member);
end;

```

```

end;

Procedure RemoveSem (ID : longint);

var S : TSemun;

begin
  If semctl (Id,0,IPC_RMID,s)<>-1 then
    Writeln ( 'Semaphore removed')
  else
    DoError ( 'Couldn't remove semaphore');
end;

Procedure ChangeMode (ID,Mode : longint);

Var rc : longint;
    opts : TSEMun;
    semds : TSEMid_ds;

begin
  opts.buf:=@semds;
  If not semctl (Id,0,IPC_STAT,opts)<>-1 then
    DoError ( 'Couldn't stat semaphore');
  Writeln ( 'Old permissions were : ',semds.sem_perm.mode);
  semds.sem_perm.mode:=mode;
  If semctl (id,0,IPC_SET,opts)<>-1 then
    Writeln ( 'Set permissions to ',mode)
  else
    DoError ( 'Couldn't set permissions');
end;

Procedure PrintSem (ID : longint);

Var l,cnt : longint;

begin
  cnt:=getmembercount(ID);
  Writeln ( 'Semaphore ',ID, ' has ',cnt, ' Members');
  For l:=0 to cnt-1 Do
    DispVal(id,i);
end;

Procedure USage;

begin
  Writeln ( 'Usage : semtool c(reate) <count>');
  Writeln ( '                l(ock) <member>');
  Writeln ( '                u(nlock) <member>');
  Writeln ( '                d(etele)');
  Writeln ( '                m(ode) <mode>');
  halt(1);
end;

Function StrToInt (S : String): longint;

Var M : longint;
    C : Integer;

```

```

begin
  val (S,M,C);
  If C<>0 Then DoError ( 'StrToInt : '+S);
  StrToInt:=M;
end;

Var Key : TKey;
    ID : Longint;

begin
  If ParamCount<1 then Usage;
  key:=ftok ( ' . ', 's' );
  Case UpCase(Paramstr(1)[1]) of
    'C' : begin
      if paramcount<>2 then usage;
      CreateSem (key , strtoint (paramstr(2)));
      end;
    'L' : begin
      if paramcount<>2 then usage;
      ID:=OpenSem (key);
      LockSem (ID , strtoint (paramstr(2)));
      end;
    'U' : begin
      if paramcount<>2 then usage;
      ID:=OpenSem (key);
      UnLockSem (ID , strtoint (paramstr(2)));
      end;
    'M' : begin
      if paramcount<>2 then usage;
      ID:=OpenSem (key);
      ChangeMode (ID , strtoint (paramstr(2)));
      end;
    'D' : Begin
      ID:=OpenSem(Key);
      RemoveSem(Id);
      end;
    'P' : begin
      ID:=OpenSem(Key);
      PrintSem(Id);
      end;
  else
    Usage
  end;
end.

```

semget

Synopsis: Return the ID of a semaphore set, possibly creating the set

Declaration: `function semget(key: TKey;nsems: cint;semflg: cint) : cint`

Visibility: default

Description: `msgget` returns the ID of the semaphore set described by `key`. Depending on the flags in `semflg`, a new queue is created.

`semflg` can have one or more of the following values (combined by ORs):

IPC_CREATThe queue is created if it doesn't already exist.

IPC_EXCLIf used in combination with **IPC_CREAT**, causes the call to fail if the set already exists. It cannot be used by itself.

Optionally, the flags can be ORed with a permission mode, which is the same mode that can be used in the file system.

if a new set of semaphores is created, then there will be `nsems` semaphores in it.

Errors: On error, -1 is returned, and `IPCError` is set.

See also: `ftok` (561), `semop` (571), `semctl` (566)

semop

Synopsis: Perform semaphore operation.

Declaration: `function semop(semid: cint;sops: PSEMbuf;nsops: cuint) : cint`

Visibility: default

Description: `semop` performs a set of operations on a message queue. `sops` points to an array of type `TSEMbuf`. The array should contain `nsops` elements.

The fields of the `TSEMbuf` (560) structure

```
TSEMbuf = record
    sem_num : word;
    sem_op  : integer;
    sem_flg : integer;
```

should be filled as follows:

sem_numThe number of the semaphore in the set on which the operation must be performed.

sem_opThe operation to be performed. The operation depends on the sign of `sem_op`: A positive number is simply added to the current value of the semaphore. If 0 (zero) is specified, then the process is suspended until the specified semaphore reaches zero. If a negative number is specified, it is subtracted from the current value of the semaphore. If the value would become negative then the process is suspended until the value becomes big enough, unless `IPC_NOWAIT` is specified in the `sem_flg`.

sem_flgOptional flags: if `IPC_NOWAIT` is specified, then the calling process will never be suspended.

The function returns `True` if the operations were successful, `False` otherwise.

Errors: In case of error, `False` is returned, and `IPCError` is set.

See also: `semget` (570), `semctl` (566)

shmat

Synopsis: Attach a shared memory block.

Declaration: `function shmat(shmid: cint;shmaddr: pointer;shmflg: cint) : pointer`

Visibility: default

Description: `shmat` attaches a shared memory block with identified `shmid` to the current process. The function returns a pointer to the shared memory block.

If `shmaddr` is `Nil`, then the system chooses a free unmapped memory region, as high up in memory space as possible.

If `shmaddr` is non-`nil`, and `SHM_RND` is in `shmflg`, then the returned address is `shmaddr`, rounded down to `SHMLBA`. If `SHM_RND` is not specified, then `shmaddr` must be a page-aligned address.

The parameter `shmflg` can be used to control the behaviour of the `shmat` call. It consists of a ORed combination of the following constants:

SHM_RNDThe suggested address in `shmaddr` is rounded down to `SHMLBA`.

SHM_RDONLYthe shared memory is attached for read access only. Otherwise the memory is attached for read-write. The process then needs read-write permissions to access the shared memory.

For an example, see `shmctl` (572).

Errors: If an error occurs, -1 is returned, and `IPCError` is set.

See also: `shmget` (574), `shmdt` (574), `shmctl` (572)

shmctl

Synopsis: Perform control operations on a shared memory block.

Declaration: `function shmctl(shmid: cint;cmd: cint;buf: PShmid_DS) : cint`

Visibility: default

Description: `shmctl` performs various operations on the shared memory block identified by identifier `shmid`.

The `buf` parameter points to a `TSHMid_ds` record. The `cmd` parameter is used to pass which operation is to be performed. It can have one of the following values :

IPC_STAT`shmctl` fills the `TSHMid_ds` record that `buf` points to with the available information about the shared memory block.

IPC_SETapplies the values in the `ipc_perm` record that `buf` points to, to the shared memory block.

IPC_RMIDthe shared memory block is destroyed (after all processes to which the block is attached, have detached from it).

If successful, the function returns `True`, `False` otherwise.

Errors: If an error occurs, the function returns `False`, and `IPCError` is set.

See also: `shmget` (574), `shmat` (571), `shmdt` (574)

Listing: `./ipcex/shmtool.pp`

```

Program shmtool;

uses ipc, strings;

Const SegSize = 100;

var key : Tkey;
```

```

    shmid,cntr : longint;
    segptr : pchar;

```

Procedure USage;

```

begin
    Writeln ( 'Usage : shmtool w(rite) text' );
    writeln ( '           r(ead)' );
    writeln ( '           d(elete)' );
    writeln ( '           m(ode change) mode' );
    halt(1);
end;

```

Procedure Writeshm (ID : Longint; **ptr** : pchar; S : **string**);

```

begin
    strcpy ( ptr ,s );
end;

```

Procedure Readshm(ID : longint; **ptr** : pchar);

```

begin
    Writeln ( 'Read : ',ptr );
end;

```

Procedure removeshm (ID : Longint);

```

begin
    shmctl (ID,IPC_RMID,Nil);
    writeln ( 'Shared memory marked for deletion' );
end;

```

Procedure CHangeMode (ID : longint; mode : **String**);

```

Var m : word;
    code : integer;
    data : TSHMid_ds;

```

```

begin
    val (mode,m,code);
    if code<>0 then
        usage;
    If Not shmctl (shmid,IPC_STAT,@data) then
        begin
            writeln ( 'Error : shmctl : ',ipccerror);
            halt(1);
        end;
    writeln ( 'Old permissions : ',data.shm_perm.mode);
    data.shm_perm.mode:=m;
    If Not shmctl (shmid,IPC_SET,@data) then
        begin
            writeln ( 'Error : shmctl : ',ipccerror);
            halt(1);
        end;
    writeln ( 'New permissions : ',data.shm_perm.mode);
end;

```

begin

```

if paramcount<1 then usage;
key := ftok ( '.', 'S' );
shmid := shmget(key, segsize, IPC_CREAT or IPC_EXCL or 438);
if shmid=-1 then
  begin
    WriteLn ( 'Shared memory exists. Opening as client' );
    shmid := shmget(key, segsize, 0);
    if shmid = -1 then
      begin
        WriteLn ( 'shmget : Error !', ipcerror );
        halt(1);
      end
    end
  else
    WriteLn ( 'Creating new shared memory segment.' );
    segptr:=shmat(shmid, nil, 0);
    if longint(segptr)=-1 then
      begin
        WriteLn ( 'Shmat : error !', ipcerror );
        halt(1);
      end;
    case upcase(paramstr(1)[1]) of
      'W' : writeshm ( shmid, segptr, paramstr(2));
      'R' : readshm ( shmid, segptr);
      'D' : removeshm(shmid);
      'M' : changemode ( shmid, paramstr(2));
    else
      begin
        writeln ( paramstr(1));
        usage;
      end;
    end;
  end;
end.

```

shmdt

Synopsis: Detach shared memory block.

Declaration: function shmdt(shmaddr: pointer) : cint

Visibility: default

Description: shmdt detaches the shared memory at address shmaddr. This shared memory block is unavailable to the current process, until it is attached again by a call to shmat ([571](#)).

The function returns True if the memory block was detached successfully, False otherwise.

Errors: On error, False is returned, and IPCError is set.

See also: shmget ([574](#)), shmat ([571](#)), shmctl ([572](#))

shmget

Synopsis: Return the ID of a shared memory block, possibly creating it

Declaration: function shmget(key: TKey;size: cint;flag: cint) : cint

Visibility: default

Description: `shmget` returns the ID of a shared memory block, described by `key`. Depending on the flags in `flag`, a new memory block is created.

`flag` can have one or more of the following values (combined by ORs):

IPC_CREAT The queue is created if it doesn't already exist.

IPC_EXCL If used in combination with `IPC_CREAT`, causes the call to fail if the queue already exists. It cannot be used by itself.

Optionally, the flags can be ORed with a permission mode, which is the same mode that can be used in the file system.

if a new memory block is created, then it will have size `Size` bytes in it.

Errors: On error, -1 is returned, and `IPCError` is set.

Chapter 15

Reference for unit 'Keyboard'

15.1 Writing a keyboard driver

Writing a keyboard driver means that hooks must be created for most of the keyboard unit functions. The `TKeyboardDriver` record contains a field for each of the possible hooks:

```
TKeyboardDriver = Record
  InitDriver : Procedure;
  DoneDriver : Procedure;
  GetKeyEvent : Function : TKeyEvent;
  PollKeyEvent : Function : TKeyEvent;
  GetShiftState : Function : Byte;
  TranslateKeyEvent : Function (KeyEvent: TKeyEvent): TKeyEvent;
  TranslateKeyEventUnicode: Function (KeyEvent: TKeyEvent): TKeyEvent;
end;
```

The meaning of these hooks is explained below:

InitDriver Called to initialize and enable the driver. Guaranteed to be called only once. This should initialize all needed things for the driver.

DoneDriver Called to disable and clean up the driver. Guaranteed to be called after a call to `initDriver`. This should clean up all things initialized by `InitDriver`.

GetKeyEvent Called by `GetKeyEvent` ([584](#)). Must wait for and return the next key event. It should NOT store keys.

PollKeyEvent Called by `PollKeyEvent` ([589](#)). It must return the next key event if there is one. Should not store keys.

GetShiftState Called by `PollShiftStateEvent` ([590](#)). Must return the current shift state.

TranslateKeyEvent Should translate a raw key event to a correct key event, i.e. should fill in the shiftstate and convert function key scancodes to function key keycodes. If the `TranslateKeyEvent` is not filled in, a default translation function will be called which converts the known scancodes from the tables in the previous section to a correct keyevent.

TranslateKeyEventUnicode Should translate a key event to a unicode key representation.

Strictly speaking, only the `GetKeyEvent` and `PollKeyEvent` hooks must be implemented for the driver to function correctly.

The example unit demonstrates how a keyboard driver can be installed. It takes the installed driver, and hooks into the `GetKeyEvent` function to register and log the key events in a file. This driver can work on top of any other driver, as long as it is inserted in the `uses` clause *after* the real driver unit, and the real driver unit should set the driver record in its initialization section.

Note that with a simple extension of this unit could be used to make a driver that is capable of recording and storing a set of keyboard strokes, and replaying them at a later time, so a 'keyboard macro' capable driver. This driver could sit on top of any other driver.

15.2 Keyboard scan codes

Special physical keys are encoded with the DOS scan codes for these keys in the second byte of the `TKeyEvent` (582) type. A complete list of scan codes can be found in the below table. This is the list of keys that is used by the default key event translation mechanism. When writing a keyboard driver, either these constants should be returned by the various key event functions, or the `TranslateKeyEvent` hook should be implemented by the driver.

A list of scan codes for special keys and combinations with the `SHIFT`, `ALT` and `CTRL` keys can be found in the following table: They are for quick reference only.

15.3 Overview

The `KeyBoard` unit implements a keyboard access layer which is system independent. It can be used to poll the keyboard state and wait for certain events. Waiting for a keyboard event can be done with the `GetKeyEvent` (584) function, which will return a driver-dependent key event. This key event can be translated to a interpretable event by the `TranslateKeyEvent` (593) function. The result of this function can be used in the other event examining functions.

A custom keyboard driver can be installed using the `SetKeyboardDriver` (592) function. The current keyboard driver can be retrieved using the `GetKeyboardDriver` (584) function. The last section of this chapter demonstrates how to make a keyboard driver.

15.4 Constants, types and variables

Constants

`AltPrefix : Byte = 0`

Alt key name index.

`CtrlPrefix : Byte = 0`

Alt key name index.

`errKbdBase = 1010`

Base of keyboard routine error reporting constants.

`errKbdInitError = errKbdBase + 0`

Failed to initialize keyboard driver

`errKbdNotImplemented = errKbdBase + 1`

Keyboard driver not implemented.

`kbAlt = 8`

Alt key modifier

`kbASCII = $00`

Ascii code key event

`kbCtrl = 4`

Control key modifier

`kbdApps = $FF17`

`kbdDelete = $FF2A`

Delete key pressed

`kbdDown = $FF27`

Arrow down key pressed

`kbdEnd = $FF26`

End key pressed

`kbdF1 = $FF01`

F1 function key pressed.

`kbdF10 = $FF0A`

F10 function key pressed.

`kbdF11 = $FF0B`

F12 function key pressed.

`kbdF12 = $FF0C`

F12 function key pressed.

`kbdF13 = $FF0D`

F13 function key pressed.

`kbdF14 = $FF0E`

F14 function key pressed.

kbdF15 = \$FF0F

F15 function key pressed.

kbdF16 = \$FF10

F16 function key pressed.

kbdF17 = \$FF11

F17 function key pressed.

kbdF18 = \$FF12

F18 function key pressed.

kbdF19 = \$FF13

F19 function key pressed.

kbdF2 = \$FF02

F2 function key pressed.

kbdF20 = \$FF14

F20 function key pressed.

kbdF3 = \$FF03

F3 function key pressed.

kbdF4 = \$FF04

F4 function key pressed.

kbdF5 = \$FF05

F5 function key pressed.

kbdF6 = \$FF06

F6 function key pressed.

kbdF7 = \$FF07

F7 function key pressed.

kbdF8 = \$FF08

F8 function key pressed.

kbdF9 = \$FF09

F9 function key pressed.

kbdHome = \$FF20

Home key pressed

kbdInsert = \$FF29

Insert key pressed

kbdLeft = \$FF23

Arrow left key pressed

kbdLWin = \$FF15

kbdMiddle = \$FF24

Middle key pad key pressed (numerical 5)

kbdPgDn = \$FF28

Page down key pressed

kbdPgUp = \$FF22

Page Up key pressed

kbdRight = \$FF25

Arrow right key pressed

kbdRWin = \$FF16

kbdUp = \$FF21

Arrow up key pressed

kbFnKey = \$02

function key pressed.

kbLeftShift = 1

Left shift key modifier

kbPhys = \$03

Physical key code event

kbReleased = \$04

Key release event

`kbRightShift = 2`

Right shift key modifier

`kbShift = kbLeftShift or kbRightShift`

Shift key modifier

`kbUnicode = $01`

Unicode code key event

`SAnd : String = 'AND'`

This constant is used as the 'And' word in key descriptions. This constant is used by the key event description routines. It can be changed to localize the key descriptions when needed.

`ShiftPrefix : Byte = 0`

Shift key name index.

`SKeyPad : Array[0..($FF2F-kbdHome)] of String = ('Home', 'Up', 'PgUp', 'Left', 'Middle',`

This constant describes all keypad keys. This constant is used by the key event description routines. It can be changed to localize the key descriptions when needed.

`SLeftRight : Array[1..2] of String = ('LEFT', 'RIGHT')`

This constant contains strings to describe left and right keys. This constant is used by the key event description routines. It can be changed to localize the key descriptions when needed.

`SScanCode : String = 'Key with scancode '`

This constant contains a string to denote a scancode key event. This constant is used by the key event description routines. It can be changed to localize the key descriptions when needed.

`SShift : Array[1..3] of String = ('SHIFT', 'CTRL', 'ALT')`

This constant describes the various modifier keys. This constant is used by the key event description routines. It can be changed to localize the key descriptions when needed.

`SUnicodeChar : String = 'Unicode character '`

This constant contains a string to denote a unicode key event. This constant is used by the key event description routines. It can be changed to localize the key descriptions when needed.

`SUnknownFunctionKey : String = 'Unknown function key : '`

This constant contains a string to denote that an unknown function key was found. This constant is used by the key event description routines. It can be changed to localize the key descriptions when needed.

Types

```
TKeyboardDriver = record
  InitDriver : procedure;
  DoneDriver : procedure;
  GetKeyEvent : function : TKeyEvent;
  PollKeyEvent : function : TKeyEvent;
  GetShiftState : function : Byte;
  TranslateKeyEvent : function(KeyEvent: TKeyEvent) : TKeyEvent;
  TranslateKeyEventUniCode : function(KeyEvent: TKeyEvent) : TKeyEvent;
end
```

The TKeyboardDriver record can be used to install a custom keyboard driver with the SetKeyboardDriver (592) function.

The various fields correspond to the different functions of the keyboard unit interface. For more information about this record see kbdriver (576)

```
TKeyEvent = Cardinal
```

The TKeyEvent type is the base type for all keyboard events.

The key stroke is encoded in the 4 bytes of the TKeyEvent type. The various fields of the key stroke encoding can be obtained by typecasting the TKeyEvent type to the TKeyRecord (582) type.

```
TKeyRecord = packed record
  KeyCode : Word;
  ShiftState : Byte;
  Flags : Byte;
end
```

The structure of a TKeyRecord structure is explained in the following table:

The shift-state can be checked using the various shift-state constants, and the flags in the last byte can be checked using one of the kbASCII, kbUniCode, kbFnKey, kbPhys, kbReleased constants.

If there are two keys returning the same char-code, there's no way to find out which one was pressed (Gray+ and Simple+). If it needs to be known which was pressed, the untranslated keycodes must be used, but these are system dependent. System dependent constants may be defined to cover those, with possibly having the same name (but different value).

15.5 Procedures and functions

AddSequence

Declaration: procedure AddSequence(const St: String; AChar: Byte; AScan: Byte)

Visibility: default

DoneKeyboard

Synopsis: Deactivate keyboard driver.

Declaration: procedure DoneKeyboard

Visibility: default

Description: DoneKeyboard de-initializes the keyboard interface if the keyboard driver is active. If the keyboard driver is not active, the function does nothing.

This will cause the keyboard driver to clear up any allocated memory, or restores the console or terminal the program was running in to its initial state before the call to InitKeyBoard (587). This function should be called on program exit. Failing to do so may leave the terminal or console window in an unusable state. Its exact action depends on the platform on which the program is running.

For an example, see most other functions.

Errors: None.

See also: InitKeyBoard (587)

FindSequence

Declaration: function FindSequence(const St: String;var AChar: Byte;var Ascan: Byte)
: Boolean

Visibility: default

FunctionKeyName

Synopsis: Return string representation of a function key code.

Declaration: function FunctionKeyName(KeyCode: Word) : String

Visibility: default

Description: FunctionKeyName returns a string representation of the function key with code KeyCode. This can be an actual function key, or one of the cursor movement keys.

Errors: In case KeyCode does not contain a function code, the SUnknownFunctionKey string is returned, appended with the KeyCode.

See also: ShiftStateToString (592), KeyEventToString (589)

Listing: ./kbdex/ex8.pp

Program Example8;

{ Program to demonstrate the FunctionKeyName function. }

Uses keyboard;

Var

K : TKeyEvent;

begin

InitKeyboard;

WriteIn('Press function keys, press "q" to end.');

Repeat

K:=GetKeyEvent;

K:=TranslateKeyEvent(K);

If IsFunctionKey(k) **then**

begin

Write('Got function key : ');

```

        WriteLn ( FunctionKeyName ( TkeyRecord (K) . KeyCode ) );
    end;
    Until ( GetKeyEventChar (K)= 'q ' );
        DoneKeyboard;
end.

```

GetKeyboardDriver

Synopsis: Return the current keyboard driver record.

Declaration: `procedure GetKeyboardDriver (var Driver: TKeyboardDriver)`

Visibility: default

Description: `GetKeyboardDriver` returns in `Driver` the currently active keyboard driver. This function can be used to enhance an existing keyboarddriver.

For more information on getting and setting the keyboard driver `kbddriver` ([576](#)).

Errors: None.

See also: `SetKeyboardDriver` ([592](#))

GetKeyEvent

Synopsis: Get the next raw key event, wait if needed.

Declaration: `function GetKeyEvent : TKeyEvent`

Visibility: default

Description: `GetKeyEvent` returns the last keyevent if one was stored in `PendingKeyEvent`, or waits for one if none is available. A non-blocking version is available in `PollKeyEvent` ([589](#)).

The returned key is encoded as a `TKeyEvent` type variable, and is normally the physical key scan code, (the scan code is driver dependent) which can be translated with one of the translation functions `TranslateKeyEvent` ([593](#)) or `TranslateKeyEventUnicode` ([593](#)). See the types section for a description of how the key is described.

Errors: If no key became available, 0 is returned.

See also: `PutKeyEvent` ([591](#)), `PollKeyEvent` ([589](#)), `TranslateKeyEvent` ([593](#)), `TranslateKeyEventUnicode` ([593](#))

Listing: `./kbdex/ex1.pp`

```

program example1;

{ This program demonstrates the GetKeyEvent function }

uses keyboard;

Var
    K : TKeyEvent;

begin
    InitKeyBoard;
    WriteLn ('Press keys , press "q" to end. ');
    Repeat

```

```

K:=GetKeyEvent;
K:=TranslateKeyEvent(K);
Write('Got key event with ');
Case GetKeyEventFlags(K) of
    kbASCII      : WriteLn('ASCII key');
    kbUnicode    : WriteLn('Unicode key');
    kbFnKey      : WriteLn('Function key');
    kbPhys       : WriteLn('Physical key');
    kbReleased   : WriteLn('Released key event');
end;
K:=TranslateKeyEvent(K);
WriteLn('Got key : ', KeyEventToString(K));
Until (GetKeyEventChar(K)='q');
DoneKeyBoard;
end.

```

GetKeyEventChar

Synopsis: Get the character key part of a key event.

Declaration: `function GetKeyEventChar(KeyEvent: TKeyEvent) : Char`

Visibility: default

Description: `GetKeyEventChar` returns the charcode part of the given `KeyEvent`, if it contains a translated character key keycode. The charcode is simply the ascii code of the character key that was pressed.

It returns the null character if the key was not a character key, but e.g. a function key.

For an example, see `GetKeyEvent` ([584](#))

Errors: None.

See also: `GetKeyEventUnicode` ([587](#)), `GetKeyEventShiftState` ([586](#)), `GetKeyEventFlags` ([586](#)), `GetKeyEventCode` ([585](#)), `GetKeyEvent` ([584](#))

GetKeyEventCode

Synopsis: Translate function key part of a key event code.

Declaration: `function GetKeyEventCode(KeyEvent: TKeyEvent) : Word`

Visibility: default

Description: `GetKeyEventCode` returns the translated function keycode part of the given `KeyEvent`, if it contains a translated function key.

If the key pressed was not a function key, the null character is returned.

Errors: None.

See also: `GetKeyEventUnicode` ([587](#)), `GetKeyEventShiftState` ([586](#)), `GetKeyEventFlags` ([586](#)), `GetKeyEventChar` ([585](#)), `GetKeyEvent` ([584](#))

Listing: `./kbdex/ex2.pp`

```

Program Example2;

{ Program to demonstrate the GetKeyEventCode function. }

Uses keyboard;

Var
    K : TKeyEvent;

begin
    InitKeyBoard;
    WriteLn('Press function keys, or press "q" to end. ');
    Repeat
        K:=GetKeyEvent;
        K:=TranslateKeyEvent(K);
        If (GetKeyEventFlags(K)<>KbfnKey) then
            WriteLn('Not a function key')
        else
            begin
                Write('Got key ( ',GetKeyEventCode(K));
                WriteLn(' ) : ',KeyEventToString(K));
            end;
        Until (GetKeyEventChar(K)= 'q ');
    DoneKeyboard;
end.

```

GetKeyEventFlags

Synopsis: Extract the flags from a key event.

Declaration: `function GetKeyEventFlags(KeyEvent: TKeyEvent) : Byte`

Visibility: default

Description: `GetKeyEventFlags` returns the flags part of the given `KeyEvent`.

For an example, see `GetKeyEvent` ([584](#))

Errors: None.

See also: `GetKeyEventUniCode` ([587](#)), `GetKeyEventShiftState` ([586](#)), `GetKeyEventCode` ([585](#)), `GetKeyEventChar` ([585](#)), `GetKeyEvent` ([584](#))

GetKeyEventShiftState

Synopsis: Return the current state of the shift keys.

Declaration: `function GetKeyEventShiftState(KeyEvent: TKeyEvent) : Byte`

Visibility: default

Description: `GetKeyEventShiftState` returns the shift-state values of the given `KeyEvent`. This can be used to detect which of the modifier keys `Shift`, `Alt` or `Ctrl` were pressed. If none were pressed, zero is returned.

Note that this function does not always return expected results; In a unix X-Term, the modifier keys do not always work.

Errors: None.

See also: [GetKeyEventUnicode \(587\)](#), [GetKeyEventFlags \(586\)](#), [GetKeyEventCode \(585\)](#), [GetKeyEventChar \(585\)](#), [GetKeyEvent \(584\)](#)

Listing: ./kbdex/ex3.pp

Program Example3;

{ Program to demonstrate the GetKeyEventShiftState function. }

Uses keyboard;

Var

 K : TKeyEvent;
 S : Byte;

begin

```
InitKeyBoard;
Write( 'Press keys combined with CTRL/SHIFT/ALT' );
WriteLn( ', or press "q" to end.' );
Repeat
    K:=GetKeyEvent;
    K:=TranslateKeyEvent(K);
    S:=GetKeyEventShiftState(K);
    If (S=0) then
        WriteLn( 'No special keys pressed' )
    else
        begin
            WriteLn( 'Detected special keys : ', ShiftStateToString(K, False));
            WriteLn( 'Got key : ', KeyEventToString(K));
        end;
    Until (GetKeyEventChar(K)= 'q' );
DoneKeyboard;
end.
```

GetKeyEventUnicode

Synopsis: Return the unicode key event.

Declaration: `function GetKeyEventUnicode(KeyEvent: TKeyEvent) : Word`

Visibility: default

Description: `GetKeyEventUnicode` returns the unicode part of the given `KeyEvent` if it contains a translated unicode character.

Errors: None.

See also: [GetKeyEventShiftState \(586\)](#), [GetKeyEventFlags \(586\)](#), [GetKeyEventCode \(585\)](#), [GetKeyEventChar \(585\)](#), [GetKeyEvent \(584\)](#)

InitKeyboard

Synopsis: Initialize the keyboard driver.

Declaration: `procedure InitKeyboard`

Visibility: default

Description: `InitKeyboard` initializes the keyboard driver. If the driver is already active, it does nothing. When the driver is initialized, it will do everything necessary to ensure the functioning of the keyboard, including allocating memory, initializing the terminal etc.

This function should be called once, before using any of the keyboard functions. When it is called, the `DoneKeyboard` (582) function should also be called before exiting the program or changing the keyboard driver with `SetKeyboardDriver` (592).

For an example, see most other functions.

Errors: None.

See also: `DoneKeyboard` (582), `SetKeyboardDriver` (592)

IsFunctionKey

Synopsis: Check whether a given event is a function key event.

Declaration: `function IsFunctionKey(KeyEvent: TKeyEvent) : Boolean`

Visibility: default

Description: `IsFunctionKey` returns `True` if the given key event in `KeyEvent` was a function key or not.

Errors: None.

See also: `GetKeyEvent` (584)

Listing: `./kbdex/ex7.pp`

```
program example1;

{ This program demonstrates the GetKeyEvent function }

uses keyboard;

Var
  K : TKeyEvent;

begin
  InitKeyBoard;
  Writeln('Press keys , press "q" to end. ');
  Repeat
    K:=GetKeyEvent;
    K:=TranslateKeyEvent(K);
    If IsFunctionKey(K) then
      Writeln('Got function key : ',KeyEventToString(K))
    else
      Writeln('not a function key. ');
  Until (GetKeyEventChar(K)= 'q ');
  DoneKeyBoard;
end.
```

KeyEventToString

Synopsis: Return a string describing the key event.

Declaration: `function KeyEventToString(KeyEvent: TKeyEvent) : String`

Visibility: default

Description: `KeyEventToString` translates the key event in `KeyEvent` to a human-readable description of the pressed key. It will use the constants described in the constants section to do so.

For an example, see most other functions.

Errors: If an unknown key is passed, the scancode is returned, prefixed with the `SScanCode` string.

See also: `FunctionKeyName` ([583](#)), `ShiftStateToString` ([592](#))

KeyPressed

Synopsis: Check event queue for key press

Declaration: `function KeyPressed : Boolean`

Visibility: default

Description: `KeyPressed` checks the keyboard event queue to see whether a key event is present, and returns `True` if a key event is available. This function simply calls `PollKeyEvent` ([589](#)) and checks for a valid result.

Errors: None.

See also: `PollKeyEvent` ([589](#)), `GetKeyEvent` ([584](#))

PollKeyEvent

Synopsis: Get next key event, but does not wait.

Declaration: `function PollKeyEvent : TKeyEvent`

Visibility: default

Description: `PollKeyEvent` checks whether a key event is available, and returns it if one is found. If no event is pending, it returns 0.

Note that this does not remove the key from the pending keys. The key should still be retrieved from the pending key events list with the `GetKeyEvent` ([584](#)) function.

Errors: None.

See also: `PutKeyEvent` ([591](#)), `GetKeyEvent` ([584](#))

Listing: `./kbdex/ex4.pp`

```
program example4;

{ This program demonstrates the PollKeyEvent function }

uses keyboard;

Var
  K : TKeyEvent;
```

```

begin
  InitKeyBoard;
  WriteLn('Press keys , press "q" to end. ');
  Repeat
    K:=PollKeyEvent;
    If k<>0 then
      begin
        K:=GetKeyEvent;
        K:=TranslateKeyEvent(K);
        writeLn;
        WriteLn('Got key : ',KeyEventToString(K));
      end
    else
      write(' ');
  Until (GetKeyEventChar(K)= 'q ');
  DoneKeyBoard;
end.

```

PollShiftStateEvent

Synopsis: Check current shift state.

Declaration: `function PollShiftStateEvent : TKeyEvent`

Visibility: default

Description: `PollShiftStateEvent` returns the current shiftstate in a keyevent. This will return 0 if there is no key event pending.

Errors: None.

See also: `PollKeyEvent` ([589](#)), `GetKeyEvent` ([584](#))

Listing: `./kbdex/ex6.pp`

```

program example6;

{ This program demonstrates the PollShiftStateEvent function }

uses keyboard;

Var
  K : TKeyEvent;

begin
  InitKeyBoard;
  WriteLn('Press keys , press "q" to end. ');
  Repeat
    K:=PollKeyEvent;
    If k<>0 then
      begin
        K:=PollShiftStateEvent;
        WriteLn('Got shift state : ',ShiftStateToString(K, False));
        // Consume the key.
        K:=GetKeyEvent;
        K:=TranslateKeyEvent(K);
      end
    end
  end
end

```

```

{   else
    write ( ' ' );
    Until ( GetKeyEventChar(K)= 'q' );
    DoneKeyBoard;
end.

```

PutKeyEvent

Synopsis: Put a key event in the event queue.

Declaration: `procedure PutKeyEvent (KeyEvent : TKeyEvent)`

Visibility: default

Description: `PutKeyEvent` adds the given `KeyEvent` to the input queue. Please note that depending on the implementation this can hold only one value, i.e. when calling `PutKeyEvent` multiple times, only the last pushed key will be remembered.

Errors: None

See also: `PollKeyEvent` ([589](#)), `GetKeyEvent` ([584](#))

Listing: `./kbdex/ex5.pp`

```

program example5;

{ This program demonstrates the PutKeyEvent function }

uses keyboard;

Var
    K, k2 : TKeyEvent;

begin
    InitKeyBoard;
    WriteLn ( 'Press keys , press "q" to end.' );
    K2:=0;
    Repeat
        K:=GetKeyEvent;
        If k<>0 then
            begin
                if (k2 mod 2)=0 then
                    K2:=K+1
                else
                    K2:=0;
                K:=TranslateKeyEvent(K);
                WriteLn ( 'Got key : ', KeyEventToString(K));
                if (K2<>0) then
                    begin
                        PutKeyEvent(k2);
                        K2:=TranslateKeyEvent(K2);
                        WriteLn ( 'Put key : ', KeyEventToString(K2))
                    end
                end
            end
        Until ( GetKeyEventChar(K)= 'q' );
        DoneKeyBoard;
    end.

```

RawReadKey

Declaration: `function RawReadKey : Char`

Visibility: default

RawReadString

Declaration: `function RawReadString : String`

Visibility: default

RestoreStartMode

Declaration: `procedure RestoreStartMode`

Visibility: default

SetKeyboardDriver

Synopsis: Set a new keyboard driver.

Declaration: `function SetKeyboardDriver(const Driver: TKeyboardDriver) : Boolean`

Visibility: default

Description: `SetKeyBoardDriver` sets the keyboard driver to `Driver`, if the current keyboard driver is not yet initialized. If the current keyboard driver is initialized, then `SetKeyboardDriver` does nothing. Before setting the driver, the currently active driver should be disabled with a call to `DoneKeyboard` (582).

The function returns `True` if the driver was set, `False` if not.

For more information on setting the keyboard driver, see `kbddriver` (576).

Errors: None.

See also: `GetKeyboardDriver` (584), `DoneKeyboard` (582)

ShiftStateToString

Synopsis: Return description of key event shift state

Declaration: `function ShiftStateToString(KeyEvent: TKeyEvent; UseLeftRight: Boolean) : String`

Visibility: default

Description: `ShiftStateToString` returns a string description of the shift state of the key event `KeyEvent`. This can be an empty string.

The shift state is described using the strings in the `SShift` constant.

For an example, see `PollShiftStateEvent` (590).

Errors: None.

See also: `FunctionKeyName` (583), `KeyEventToString` (589)

TranslateKeyEvent

Synopsis: Translate raw event to ascii key event

Declaration: `function TranslateKeyEvent(KeyEvent: TKeyEvent) : TKeyEvent`

Visibility: default

Description: `TranslateKeyEvent` performs ASCII translation of the `KeyEvent`. It translates a physical key to a function key if the key is a function key, and translates the physical key to the ordinal of the ascii character if there is an equivalent character key.

For an example, see `GetKeyEvent` ([584](#))

Errors: None.

See also: `TranslateKeyEventUnicode` ([593](#))

TranslateKeyEventUnicode

Synopsis: Translate raw event to UNICODE key event

Declaration: `function TranslateKeyEventUnicode(KeyEvent: TKeyEvent) : TKeyEvent`

Visibility: default

Description: `TranslateKeyEventUnicode` performs Unicode translation of the `KeyEvent`. It is not yet implemented for all platforms.

Errors: If the function is not yet implemented, then the `ErrorCode` of the `system` unit will be set to `errKbdNotImplemented`

See also: `TranslateKeyEvent` ([593](#))

Table 15.1: Key Scancodes

Code	Key	Code	Key	Code	Key
00	NoKey	3D	F3	70	ALT-F9
01	ALT-Esc	3E	F4	71	ALT-F10
02	ALT-Space	3F	F5	72	CTRL-PrtSc
04	CTRL-Ins	40	F6	73	CTRL-Left
05	SHIFT-Ins	41	F7	74	CTRL-Right
06	CTRL-Del	42	F8	75	CTRL-end
07	SHIFT-Del	43	F9	76	CTRL-PgDn
08	ALT-Back	44	F10	77	CTRL-Home
09	ALT-SHIFT-Back	47	Home	78	ALT-1
0F	SHIFT-Tab	48	Up	79	ALT-2
10	ALT-Q	49	PgUp	7A	ALT-3
11	ALT-W	4B	Left	7B	ALT-4
12	ALT-E	4C	Center	7C	ALT-5
13	ALT-R	4D	Right	7D	ALT-6
14	ALT-T	4E	ALT-GrayPlus	7E	ALT-7
15	ALT-Y	4F	end	7F	ALT-8
16	ALT-U	50	Down	80	ALT-9
17	ALT-I	51	PgDn	81	ALT-0
18	ALT-O	52	Ins	82	ALT-Minus
19	ALT-P	53	Del	83	ALT-Equal
1A	ALT-LftBrack	54	SHIFT-F1	84	CTRL-PgUp
1B	ALT-RgtBrack	55	SHIFT-F2	85	F11
1E	ALT-A	56	SHIFT-F3	86	F12
1F	ALT-S	57	SHIFT-F4	87	SHIFT-F11
20	ALT-D	58	SHIFT-F5	88	SHIFT-F12
21	ALT-F	59	SHIFT-F6	89	CTRL-F11
22	ALT-G	5A	SHIFT-F7	8A	CTRL-F12
23	ALT-H	5B	SHIFT-F8	8B	ALT-F11
24	ALT-J	5C	SHIFT-F9	8C	ALT-F12
25	ALT-K	5D	SHIFT-F10	8D	CTRL-Up
26	ALT-L	5E	CTRL-F1	8E	CTRL-Minus
27	ALT-SemiCol	5F	CTRL-F2	8F	CTRL-Center
28	ALT-Quote	60	CTRL-F3	90	CTRL-GreyPlus
29	ALT-OpQuote	61	CTRL-F4	91	CTRL-Down
2B	ALT-BkSlash	62	CTRL-F5	94	CTRL-Tab
2C	ALT-Z	63	CTRL-F6	97	ALT-Home
2D	ALT-X	64	CTRL-F7	98	ALT-Up
2E	ALT-C	65	CTRL-F8	99	ALT-PgUp
2F	ALT-V	66	CTRL-F9	9B	ALT-Left
30	ALT-B	67	CTRL-F10	9D	ALT-Right
31	ALT-N	68	ALT-F1	9F	ALT-end
32	ALT-M	69	ALT-F2	A0	ALT-Down
33	ALT-Comma	6A	ALT-F3	A1	ALT-PgDn
34	ALT-Period	6B	ALT-F4	A2	ALT-Ins
35	ALT-Slash	6C	ALT-F5	A3	ALT-Del
37	ALT-GreyAst	6D	ALT-F6	A5	ALT-Tab
3B	F1	6E	ALT-F7		
3C	F2	6F	ALT-F8		

Table 15.2: Special keys scan codes

Key	Code	SHIFT-Key	CTRL-Key	Alt-Key
NoKey	00			
F1	3B	54	5E	68
F2	3C	55	5F	69
F3	3D	56	60	6A
F4	3E	57	61	6B
F5	3F	58	62	6C
F6	40	59	63	6D
F7	41	5A	64	6E
F8	42	5A	65	6F
F9	43	5B	66	70
F10	44	5C	67	71
F11	85	87	89	8B
F12	86	88	8A	8C
Home	47		77	97
Up	48		8D	98
PgUp	49		84	99
Left	4B		73	9B
Center	4C		8F	
Right	4D		74	9D
end	4F		75	9F
Down	50		91	A0
PgDn	51		76	A1
Ins	52	05	04	A2
Del	53	07	06	A3
Tab	8	0F	94	A5
GreyPlus			90	4E

Table 15.3: Structure of TKeyRecord

Field	Meaning
KeyCode	Depending on flags either the physical representation of a key (under DOS scancode, ascii code pair), or the t
ShiftState	Shift-state when this key was pressed (or shortly after)
Flags	Determine how to interpret KeyCode

Chapter 16

Reference for unit 'Linux'

16.1 Overview

The linux unit contains linux specific operating system calls.

The platform independent functionality of the FPC 1.0.X version of the linux unit has been split out over the unix ([1234](#)), baseunix ([71](#)) and unixutil ([1282](#)) units.

The X86-specific parts have been moved to the X86 ([1309](#)) unit.

People wanting to use the old version (FPC 1.0.X and before) of the linux can use the oldlinux ([737](#)) unit instead.

16.2 Constants, types and variables

Constants

`CLONE_FILES = $00000400`

Clone ([597](#)) option: open files shared between processes

`CLONE_FS = $00000200`

Clone ([597](#)) option: fs info shared between processes

`CLONE_PID = $00001000`

Clone ([597](#)) option: PID shared between processes

`CLONE_SIGHAND = $00000800`

Clone ([597](#)) option: signal handlers shared between processes

`CLONE_VM = $00000100`

Clone ([597](#)) option: VM shared between processes

`CSIGNAL = $000000ff`

Clone ([597](#)) option: Signal mask to be sent at exit

Types

`PSysInfo = ^TSysinfo`

Pointer to `TSysInfo` (597) record.

`TCloneFunc = function(args: pointer) : LongInt`

Clone function prototype.

```
TSysinfo = packed record
  uptime : LongInt;
  loads : Array[1..3] of LongInt;
  totalram : LongInt;
  freeram : LongInt;
  sharedram : LongInt;
  bufferram : LongInt;
  totalswap : LongInt;
  freeswap : LongInt;
  procs : Integer;
  s : String;
end
```

Record with system information, used by the `SysInfo` (599) call.

16.3 Procedures and functions

Clone

Synopsis: Clone current process (create new thread)

Declaration: `function Clone(func: TCloneFunc; sp: pointer; flags: LongInt; args: pointer) : LongInt`

Visibility: default

Description: `Clone` creates a child process which is a copy of the parent process, just like `FpFork` (109) does. In difference with `Fork`, however, the child process shares some parts of its execution context with its parent, so it is suitable for the implementation of threads: many instances of a program that share the same memory.

When the child process is created, it starts executing the function `Func`, and passes it `Args`. The return value of `Func` is either the explicit return value of the function, or the exit code of the child process.

The `sp` pointer points to the memory reserved as stack space for the child process. This address should be the top of the memory block to be used as stack.

The `Flags` determine the behaviour of the `Clone` call. The low byte of the `Flags` contains the number of the signal that will be sent to the parent when the child dies. This may be bitwise OR'ed with the following constants:

CLONE_VMParent and child share the same memory space, including memory (un)mapped with subsequent `mmap` calls.

CLONE_FSParent and child have the same view of the filesystem; the `chroot`, `chdir` and `umask` calls affect both processes.

CLONE_FILESthe file descriptor table of parent and child is shared.

CLONE_SIGHANDthe parent and child share the same table of signal handlers. The signal masks are different, though.

CLONE_PIDParent and child have the same process ID.

Clone returns the process ID in the parent process, and -1 if an error occurred.

Errors: On error, -1 is returned to the parent, and no child is created.

sys_eagainToo many processes are running.

sys_enomemNot enough memory to create child process.

See also: `#rtl.baseunix.FpFork` ([109](#))

Listing: `./linuxex/ex71.pp`

```

program TestC{Ione};

{$ifdef Linux}
// close is very Linux specific. 1.9.x threading is done via pthreads.

uses
  Linux , Errors , crt;

const
  Ready : Boolean = false;
  aChar : Char    = 'a';

function CloneProc( Arg: Pointer ): LongInt; Cdecl;
begin
  WriteLn( 'Hello from the clone ',PChar(Arg));
  repeat
    Write(aChar);
    Select(0,0,0,0,600);
  until Ready;
  WriteLn( 'Clone finished. ');
  CloneProc := 1;
end;

var
  PID : LongInt;

procedure MainProc;
begin
  WriteLn( 'cloned process PID: ', PID );
  WriteLn( 'Press <ESC> to kill ... ' );
  repeat
    Write( '. ');
    Select(0,0,0,0,300);
    if KeyPressed then
      case ReadKey of
        #27: Ready := true;
        'a': aChar := 'A';
        'A': aChar := 'a';
        'b': aChar := 'b';
        'B': aChar := 'B';
      end;
  until Ready;

```

```

    WriteLn( 'Ready. ' );
end;

const
    StackSize = 16384;
    theFlags = CLONE_VM+CLONE_FS+CLONE_FILES+CLONE_SIGHAND;
    aMsg      : PChar = 'Oops !';

var
    theStack : Pointer;
    ExitStat : LongInt;

begin
    GetMem( theStack , StackSize );
    PID := Clone( @CloneProc,
                  Pointer( LongInt( theStack )+StackSize ),
                  theFlags ,
                  aMsg );
    if PID < 0 then
        WriteLn( 'Error : ', LinuxError , ' when cloning. ' )
    else
        begin
            MainProc;
            case WaitPID( 0 , @ExitStat , Wait_Untraced or wait_clone ) of
                -1: WriteLn( 'error: ', LinuxError , ' ; ', StrError( LinuxError ) );
                0: WriteLn( 'error: ', LinuxError , ' ; ', StrError( LinuxError ) );
            else
                WriteLn( 'Clone exited with: ', ExitStat shr 8 );
            end;
        end;
    FreeMem( theStack , StackSize );
{$else}
begin
{$endif}
end.

```

Sysinfo

Synopsis: Return kernel system information

Declaration: function Sysinfo(var Info: TSysinfo) : Boolean

Visibility: default

Description: SysInfo returns system information in Info. Returned information in Info includes:

- uptime** Number of seconds since boot.
- loads** 1, 5 and 15 minute load averages.
- totalram** total amount of main memory.
- freeram** amount of free memory.
- sharedram** amount of shared memory.
- bufferram** amount of memory used by buffers.
- totalswap** total amount of swapspace.
- freeswap** amount of free swapspace.

procs number of current processes.

Errors: None.

See also: #rtl.baseunix.fpUname ([145](#))

Listing: ./linuxex/ex64.pp

program Example64;

*{ Example to demonstrate the SysInfo function.
Sysinfo is Linux-only. }*

{ \$ifdef Linux }

Uses Linux;

Function Mb(L : Longint) : longint;

begin

 Mb:=L **div** (1024*1024);

end;

Var Info : TSysInfo;

 D,M,Secs,H : longint;

{ \$endif }

begin

{ \$ifdef Linux }

If Not SysInfo(Info) **then**

Halt(1);

With Info **do**

begin

 D:=Uptime **div** (3600*24);

 UpTime:=UpTime **mod** (3600*24);

 h:=uptime **div** 3600;

 uptime:=uptime **mod** 3600;

 m:=uptime **div** 60;

 secs:=uptime **mod** 60;

Writeln('Uptime : ',d,'days', ' ',h,' hours', ' ',m,' min', ' ',secs,' s. ');

Writeln('Loads : ',Loads[1], ' / ',Loads[2], ' / ',Loads[3]);

Writeln('Total Ram : ',Mb(totalram),'Mb. ');

Writeln('Free Ram : ',Mb(freeram),'Mb. ');

Writeln('Shared Ram : ',Mb(sharedram),'Mb. ');

Writeln('Buffer Ram : ',Mb(bufferram),'Mb. ');

Writeln('Total Swap : ',Mb(totalswap),'Mb. ');

Writeln('Free Swap : ',Mb(freeswap),'Mb. ');

end;

{ \$endif }

end.

Chapter 17

Reference for unit 'math'

17.1 Geometrical functions

Table 17.1:

Name	Description
hypot (617)	Hypotenuse of triangle
norm (629)	Euclidian norm

17.2 Statistical functions

Table 17.2:

Name	Description
mean (624)	Mean of values
meanandstddev (625)	Mean and standard deviation of values
momentskewkurtosis (628)	Moments, skew and kurtosis
popnstddev (630)	Population standarddeviation
popnvariance (630)	Population variance
randg (633)	Gaussian distributed random value
stddev (637)	Standard deviation
sum (637)	Sum of values
sumofsquares (638)	Sum of squared values
sumsandsquares (639)	Sum of values and squared values
totalvariance (641)	Total variance of values
variance (641)	variance of values

Table 17.3:

Name	Description
<code>ceil</code> (611)	Round to infinity
<code>floor</code> (614)	Round to minus infinity
<code>frexp</code> (615)	Return mantissa and exponent

17.3 Number converting

17.4 Exponential and logarithmic functions

Table 17.4:

Name	Description
<code>intpower</code> (618)	Raise float to integer power
<code>ldexp</code> (620)	Calculate $2^p \times x$
<code>lnxp1</code> (620)	calculate $\log(x+1)$
<code>log10</code> (621)	calculate 10-base log
<code>log2</code> (621)	calculate 2-base log
<code>logn</code> (622)	calculate N-base log
<code>power</code> (631)	raise float to arbitrary power

17.5 Hyperbolic functions

Table 17.5:

Name	Description
<code>arcosh</code> (607)	calculate reverse hyperbolic cosine
<code>arsinh</code> (610)	calculate reverse hyperbolic sine
<code>artanh</code> (610)	calculate reverse hyperbolic tangent
<code>cosh</code> (611)	calculate hyperbolic cosine
<code>sinh</code> (636)	calculate hyperbolic sine
<code>tanh</code> (640)	calculate hyperbolic tangent

17.6 Trigonometric functions

17.7 Angle unit conversion

Routines to convert angles between different angle units.

Table 17.6:

Name	Description
<code>arccos</code> (606)	calculate reverse cosine
<code>arcsin</code> (608)	calculate reverse sine
<code>arctan2</code> (609)	calculate reverse tangent
<code>cotan</code> (612)	calculate cotangent
<code>sincos</code> (635)	calculate sine and cosine
<code>tan</code> (640)	calculate tangent

Table 17.7:

Name	Description
<code>cycleto rad</code> (612)	convert cycles to radians
<code>degtograd</code> (613)	convert degrees to grads
<code>degtorad</code> (613)	convert degrees to radians
<code>gradtodeg</code> (616)	convert grads to degrees
<code>gradtorad</code> (617)	convert grads to radians
<code>radto cycle</code> (632)	convert radians to cycles
<code>radtodeg</code> (632)	convert radians to degrees
<code>radto grad</code> (633)	convert radians to grads

17.8 Min/max determination

Functions to determine the minimum or maximum of numbers:

Table 17.8:

Name	Description
<code>max</code> (622)	Maximum of 2 values
<code>maxIntValue</code> (623)	Maximum of an array of integer values
<code>maxvalue</code> (624)	Maximum of an array of values
<code>min</code> (626)	Minimum of 2 values
<code>minIntValue</code> (627)	Minimum of an array of integer values
<code>minvalue</code> (627)	Minimum of an array of values

17.9 Used units

17.10 Overview

This document describes the `math` unit. The `math` unit was initially written by Florian Klaempfl. It provides mathematical functions which aren't covered by the system unit.

This chapter starts out with a definition of all types and constants that are defined, after which an overview is presented of the available functions, grouped by category, and the last part contains a complete explanation of each function.

The following things must be taken into account when using this unit:

Table 17.9: Used units by unit 'math'

Name	Page
sysutils	1071

1. This unit is compiled in Object Pascal mode so all `integers` are 32 bit.
2. Some overloaded functions exist for data arrays of integers and floats. When using the address operator (`@`) to pass an array of data to such a function, make sure the address is typecasted to the right type, or turn on the 'typed address operator' feature. failing to do so, will cause the compiler not be able to decide which function you want to call.

17.11 Constants, types and variables

Constants

`EqualsValue = 0`

Values are the same

`GreaterThanValue = High (TValueRelationship)`

First values is greater than second value

`Infinity = 1.0 / 0.0`

Value is infinity

`LessThanValue = Low (TValueRelationship)`

First value is less than second value

`MaxExtended = 1.1e + 4932`

Maximum value of extended type

`MaxFloat = MaxExtended`

Maximum value of float type

`MinExtended = 3.4e - 4932`

Minimum value (closest to zero) of extended type

`MinFloat = MinExtended`

Minimum value (closest to zero) of float type

`NaN = 0.0 / 0.0`

Value is Not a Number

NegativeValue = Low (TValueSign)

Value is negative

PositiveValue = High (TValueSign)

Value is positive

ZeroValue = 0

Value is zero

Types

float = extended

All calculations are done with the Float type. This allows to recompile the unit with a different float type to obtain a desired precision. The pointer type PFloat (605) is used in functions that accept an array of values of arbitrary length.

PFloat = ^float

Pointer to Float (605) type.

PInteger = ^Integer

Pointer to integer type

TFPUException = (exInvalidOp,exDenormalized,exZeroDivide,exOverflow,
exUnderflow,exPrecision)

Table 17.10: Enumeration values for type TFPUException

Value	Explanation
exDenormalized	
exInvalidOp	Invalid operation error
exOverflow	Float overflow error
exPrecision	Precision error
exUnderflow	Float underflow error
exZeroDivide	Division by zero error.

Type describing Floating Point processor exceptions.

TFPUExceptionMask= Set of (exDenormalized,exInvalidOp,exOverflow,
exPrecision,exUnderflow,exZeroDivide)

Type to set the Floating Point Unit exception mask.

TFPUPrecisionMode = (pmSingle,pmReserved,pmDouble,pmExtended)

Type describing the default precision for the Floating Point processor.

Table 17.11: Enumeration values for type `TFPUPrecisionMode`

Value	Explanation
<code>pmDouble</code>	Double-type precision
<code>pmExtended</code>	Extended-type precision
<code>pmReserved</code>	?
<code>pmSingle</code>	Single-type precision

Table 17.12: Enumeration values for type `TFPURoundingMode`

Value	Explanation
<code>rmDown</code>	Round to biggest integer smaller than value.
<code>rmNearest</code>	Round to nearest integer value
<code>rmTruncate</code>	Cut off fractional part.
<code>rmUp</code>	Round to smallest integer larger than value.

```
TFPURoundingMode = (rmNearest,rmDown,rmUp,rmTruncate)
```

Type describing the rounding mode for the Floating Point processor.

```
tpaymenttime = (ptendofperiod,ptstartofperiod)
```

Type used in financial (interest) calculations.

```
TValueRelationship = -1..1
```

Type to describe relational order between values

```
TValueSign = -1..1
```

Type indicating sign of a valuea

17.12 Procedures and functions

arccos

Synopsis: Return inverse cosine

Declaration: `function arccos(x: float) : float`

Visibility: default

Description: `Arccos` returns the inverse cosine of its argument `x`. The argument `x` should lie between -1 and 1 (borders included).

Errors: If the argument `x` is not in the allowed range, an `EInvalidArgument` exception is raised.

See also: `arcsin` ([608](#)), `arcosh` ([607](#)), `arsinh` ([610](#)), `artanh` ([610](#))

Listing: `./mathex/ex1.pp`

Table 17.13: Enumeration values for type `tpaymenttime`

Value	Explanation
<code>ptendofperiod</code>	End of period.
<code>ptstartofperiod</code>	Start of period.

Program Example1;

{ Program to demonstrate the arccos function. }

Uses math;

Procedure WriteRadDeg(X : float);

begin

Writeln(X:8:5, ' rad = ', radtodeg(x):8:5, ' degrees.')

end;

begin

WriteRadDeg (arccos(1));

WriteRadDeg (arccos(**sqr**t(3)/2));

WriteRadDeg (arccos(**sqr**t(2)/2));

WriteRadDeg (arccos(1/2));

WriteRadDeg (arccos(0));

WriteRadDeg (arccos(-1));

end.

arccosh

Synopsis: Return inverse hyperbolic cosine

Declaration: `function arccosh(x: float) : float`

Visibility: default

Description: `arccosh` returns the inverse hyperbolic cosine of its argument `x`.

This function is an alias for `arcosh` (607), provided for Delphi compatibility.

See also: `arcosh` (607)

arcosh

Synopsis: Return inverse hyperbolic cosine

Declaration: `function arcosh(x: float) : float`

Visibility: default

Description: `Arcosh` returns the inverse hyperbolic cosine of its argument `x`. The argument `x` should be larger than 1. The `arccosh` variant of this function is supplied for Delphi compatibility.

Errors: If the argument `x` is not in the allowed range, an `EInvalidArgument` exception is raised.

See also: `cosh` (611), `sinh` (636), `arcsin` (608), `arsinh` (610), `artanh` (610), `tanh` (640)

Listing: ./mathex/ex3.pp

```

Program Example3;

{ Program to demonstrate the arcosh function . }

Uses math;

begin
  WriteLn( arcosh ( 1 ) );
  WriteLn( arcosh ( 2 ) );
end.

```

arcsin

Synopsis: Return inverse sine

Declaration: `function arcsin(x: float) : float`

Visibility: default

Description: Arcsin returns the inverse sine of its argument x. The argument x should lie between -1 and 1.

Errors: If the argument x is not in the allowed range, an `EInvalidArgument` exception is raised.

See also: `arccos` ([606](#)), `arcosh` ([607](#)), `arsinh` ([610](#)), `artanh` ([610](#))

Listing: ./mathex/ex2.pp

```

Program Example1;

{ Program to demonstrate the arcsin function . }

Uses math;

Procedure WriteRadDeg(X : float);

begin
  WriteLn(X:8:5, ' rad = ', radtodeg(x):8:5, ' degrees. ')
end;

begin
  WriteRadDeg ( arcsin ( 1 ) );
  WriteRadDeg ( arcsin ( sqrt ( 3 ) / 2 ) );
  WriteRadDeg ( arcsin ( sqrt ( 2 ) / 2 ) );
  WriteRadDeg ( arcsin ( 1 / 2 ) );
  WriteRadDeg ( arcsin ( 0 ) );
  WriteRadDeg ( arcsin ( - 1 ) );
end.

```

arsinh

Synopsis: Return inverse hyperbolic sine

Declaration: `function arcsinh(x: float) : float`

Visibility: default

Description: `arcsinh` returns the inverse hyperbolic sine of it's argument `x`.

This function is an alias for `arsinh` (610), provided for Delphi compatibility.

See also: `arsinh` (610)

arctan2

Synopsis: Return arctangent of (`y/x`)

Declaration: `function arctan2(y: float;x: float) : float`

Visibility: default

Description: `arctan2` calculates `arctan(y/x)`, and returns an angle in the correct quadrant. The returned angle will be in the range $-\pi$ to π radians. The values of `x` and `y` must be between -2^{64} and 2^{64} , moreover `x` should be different from zero. On Intel systems this function is implemented with the native intel `fpatan` instruction.

Errors: If `x` is zero, an overflow error will occur.

See also: `arccos` (606), `arcosh` (607), `arsinh` (610), `artanh` (610)

Listing: `./mathex/ex6.pp`

Program Example6;

{ Program to demonstrate the arctan2 function. }

Uses math;

Procedure WriteRadDeg(`X : float`);

begin

WriteLn(`X:8:5`, ' rad = ', `radtodeg(x):8:5`, ' degrees. ')

end;

begin

WriteRadDeg (`arctan2(1,1)`);

end.

artanh

Synopsis: Return inverse hyperbolic tangent

Declaration: `function artanh(x: float) : float`

Visibility: default

Description: `arcsinh` returns the inverse hyperbolic tangent of it's argument `x`.

This function is an alias for `artanh` (610), provided for Delphi compatibility.

See also: `artanh` (610)

arsinh

Synopsis: Return inverse hyperbolic sine

Declaration: `function arsinh(x: float) : float`

Visibility: default

Description: `arsinh` returns the inverse hyperbolic sine of its argument `x`. The `arscsinh` variant of this function is supplied for Delphi compatibility.

Errors: None.

See also: `arcosh` ([607](#)), `arccos` ([606](#)), `arcsin` ([608](#)), `artanh` ([610](#))

Listing: `./mathex/ex4.pp`

Program Example4;

{ Program to demonstrate the arsinh function. }

Uses math;

begin

WriteLn(`arsinh(0)`);

WriteLn(`arsinh(1)`);

end.

artanh

Synopsis: Return inverse hyperbolic tangent

Declaration: `function artanh(x: float) : float`

Visibility: default

Description: `artanh` returns the inverse hyperbolic tangent of its argument `x`, where `x` should lie in the interval `[-1,1]`, borders included. The `arctanh` variant of this function is supplied for Delphi compatibility.

Errors: In case `x` is not in the interval `[-1,1]`, an `EInvalidArgument` exception is raised.

See also: `arcosh` ([607](#)), `arccos` ([606](#)), `arcsin` ([608](#)), `artanh` ([610](#))

Listing: `./mathex/ex5.pp`

Program Example5;

{ Program to demonstrate the artanh function. }

Uses math;

begin

WriteLn(`artanh(0)`);

WriteLn(`artanh(0.5)`);

end.

ceil

Synopsis: Return the lowest integer number greater than or equal to argument

Declaration: `function ceil(x: float) : Integer`

Visibility: default

Description: `Ceil` returns the lowest integer number greater than or equal to `x`. The absolute value of `x` should be less than `maxint`.

Errors: If the absolute value of `x` is larger than `maxint`, an overflow error will occur.

See also: `floor` ([614](#))

Listing: `./mathex/ex7.pp`

Program `Example7`;

{ Program to demonstrate the Ceil function. }

Uses `math`;

begin

`WriteLn (Ceil (-3.7)); // should be -3`

`WriteLn (Ceil (3.7)); // should be 4`

`WriteLn (Ceil (-4.0)); // should be -4`

end.

ClearExceptions

Synopsis: Clear Floating Point Unit exceptions

Declaration: `procedure ClearExceptions(RaisePending: Boolean)`

Visibility: default

Description: Clear Floating Point Unit exceptions

cosh

Synopsis: Return hyperbolic cosine

Declaration: `function cosh(x: float) : float`

Visibility: default

Description: `Cosh` returns the hyperbolic cosine of its argument `{x}`.

Errors: None.

See also: `arcosh` ([607](#)), `sinh` ([636](#)), `arsinh` ([610](#))

Listing: `./mathex/ex8.pp`

```

Program Example8;

{ Program to demonstrate the cosh function . }

Uses math;

begin
    WriteLn (Cosh (0));
    WriteLn (Cosh (1));
end .

```

cotan

Synopsis: Return cotangent

Declaration: `function cotan(x: float) : float`

Visibility: default

Description: Cotan returns the cotangent of it's argument x. x should be different from zero.

Errors: If x is zero then a overflow error will occur.

See also: [tanh \(640\)](#)

Listing: `./mathex/ex9.pp`

```

Program Example9;

{ Program to demonstrate the cotan function . }

Uses math;

begin
    writeln (cotan (pi / 2));
    WriteLn (cotan (pi / 3));
    WriteLn (cotan (pi / 4));
end .

```

cycletorad

Synopsis: Convert cycle angle to radians angle

Declaration: `function cycletorad(cycle: float) : float`

Visibility: default

Description: Cycletorad transforms it's argument cycle (an angle expressed in cycles) to radians. (1 cycle is 2π radians).

Errors: None.

See also: [degtograd \(613\)](#), [degtorad \(613\)](#), [radto deg \(632\)](#), [radto grad \(633\)](#), [radto cycle \(632\)](#)

Listing: `./mathex/ex10.pp`

Program Example10;

{ Program to demonstrate the cycletorad function. }

Uses math;

begin

writeln(cos(cycletorad(1/6))); *// Should print 1/2*

writeln(cos(cycletorad(1/8))); *// should be sqrt(2)/2*

end.

degtograd

Synopsis: Convert degree angle to grads angle

Declaration: function degtograd(deg: float) : float

Visibility: default

Description: Degtograd transforms it's argument deg (an angle in degrees) to grads. (90 degrees is 100 grad.)

Errors: None.

See also: cycletorad ([612](#)), degtorad ([613](#)), radtodeg ([632](#)), radtograd ([633](#)), radtcycle ([632](#))

Listing: ./mathex/ex11.pp

Program Example11;

{ Program to demonstrate the degtograd function. }

Uses math;

begin

writeln(degtograd(90));

writeln(degtograd(180));

writeln(degtograd(270))

end.

degtorad

Synopsis: Convert degree angle to radians angle.

Declaration: function degtorad(deg: float) : float

Visibility: default

Description: Degtorad converts it's argument deg (an angle in degrees) to radians. (pi radians is 180 degrees)

Errors: None.

See also: cycletorad ([612](#)), degtograd ([613](#)), radtodeg ([632](#)), radtograd ([633](#)), radtcycle ([632](#))

Listing: ./mathex/ex12.pp

Program Example12;

{ Program to demonstrate the degtorad function. }

Uses math;

begin

writeln(degtorad(45));

writeln(degtorad(90));

writeln(degtorad(180));

writeln(degtorad(270));

writeln(degtorad(360));

end.

DivMod

Synopsis: Return DIV and MOD of arguments

Declaration: `procedure DivMod(Dividend: Integer; Divisor: Word; var Result: Word;
 var Remainder: Word)`

Visibility: default

Description: DivMod returns Dividend DIV Divisor in Result, and Dividend MOD Divisor in Remainder

EnsureRange

Synopsis: Change value to it falls in specified range.

Declaration: `function EnsureRange(const AValue: Integer; const AMin: Integer;
 const AMax: Integer) : Integer
function EnsureRange(const AValue: Int64; const AMin: Int64;
 const AMax: Int64) : Int64`

Visibility: default

Description: EnsureRange returns Value if AValue is in the range AMin..AMax. It returns AMin if the value is less than AMin, or AMax if the value is larger than AMax.

See also: InRange ([618](#))

floor

Synopsis: Return the largest integer smaller than or equal to argument

Declaration: `function floor(x: float) : Integer`

Visibility: default

Description: Floor returns the largest integer smaller than or equal to x. The absolute value of x should be less than maxint.

Errors: If x is larger than maxint, an overflow will occur.

See also: ceil ([611](#))

Listing: ./mathex/ex13.pp

Program Example13;

{ Program to demonstrate the floor function . }

Uses math;

begin

WriteLn(Ceil(-3.7)); *// should be -4*

WriteLn(Ceil(3.7)); *// should be 3*

WriteLn(Ceil(-4.0)); *// should be -4*

end.

Frexp

Synopsis: Return mantissa and exponent.

Declaration: `procedure Frexp(X: float; var Mantissa: float; var Exponent: Integer)`

Visibility: default

Description: `Frexp` returns the mantissa and exponent of its argument `x` in mantissa and exponent.

Errors: None

Listing: ./mathex/ex14.pp

Program Example14;

{ Program to demonstrate the frexp function . }

Uses math;

Procedure dofexp(**Const** X : extended);

var man : extended;

exp: longint;

begin

 man:=0;

exp:=0;

 frexp(x,man,**exp**);

write(x, ' has ');

WriteLn('mantissa ',man,' and exponent ',**exp**);

end;

begin

// dofexp(1.00);

 dofexp(1.02e-1);

 dofexp(1.03e-2);

 dofexp(1.02e1);

 dofexp(1.03e2);

end.

GetExceptionMask

Synopsis: Get the Floating Point Unit exception mask.

Declaration: `function GetExceptionMask : TFPUExceptionMask`

Visibility: default

Description: Get the Floating Point Unit exception mask.

GetPrecisionMode

Synopsis: Return the Floating Point Unit precision mode.

Declaration: `function GetPrecisionMode : TFPUPrecisionMode`

Visibility: default

Description: Return the Floating Point Unit precision mode.

GetRoundMode

Synopsis: Return the Floating Point Unit rounding mode.

Declaration: `function GetRoundMode : TFPURoundingMode`

Visibility: default

Description: Return the Floating Point Unit rounding mode.

GetSSECSR

Declaration: `function GetSSECSR : dword`

Visibility: default

gradtodeg

Synopsis: Convert grads angle to degrees angle

Declaration: `function gradtodeg(grad: float) : float`

Visibility: default

Description: Gradtodeg converts its argument grad (an angle in grads) to degrees. (100 grad is 90 degrees)

Errors: None.

See also: [cycletorad \(612\)](#), [degtograd \(613\)](#), [radtodeg \(632\)](#), [radtograd \(633\)](#), [radtocycle \(632\)](#), [gradtorad \(617\)](#)

Listing: `./mathex/ex15.pp`

Program Example15;

{ Program to demonstrate the gradtodeg function. }

Uses math;

begin
 writeln (gradtodeg(100));
 writeln (gradtodeg(200));
 writeln (gradtodeg(300));
end.

gradtorad

Synopsis: Convert grads angle to radians angle

Declaration: `function gradtorad(grad: float) : float`

Visibility: default

Description: Gradtorad converts its argument grad (an angle in grads) to radians. (200 grad is pi degrees).

Errors: None.

See also: cycletorad ([612](#)), degtograd ([613](#)), radtodeg ([632](#)), radtograd ([633](#)), radtcycle ([632](#)), gradtodeg ([616](#))

Listing: ./mathex/ex16.pp

Program Example16;

{ Program to demonstrate the gradtorad function. }

Uses math;

begin
 writeln (gradtorad(100));
 writeln (gradtorad(200));
 writeln (gradtorad(300));
end.

hypot

Synopsis: Return hypotenuse of triangle

Declaration: `function hypot(x: float;y: float) : float`

Visibility: default

Description: Hypot returns the hypotenuse of the triangle where the sides adjacent to the square angle have lengths x and y. The function uses Pythagoras' rule for this.

Errors: None.

Listing: ./mathex/ex17.pp

```

Program Example17;

{ Program to demonstrate the hypot function. }

Uses math;

begin
  WriteLn(hypot(3,4)); // should be 5
end.

```

InRange

Synopsis: Check whether value is in range.

Declaration: `function InRange(const AValue: Integer;const AMin: Integer; const AMax: Integer) : Boolean`
`function InRange(const AValue: Int64;const AMin: Int64; const AMax: Int64) : Boolean`

Visibility: default

Description: `InRange` returns `True` if `AValue` is in the range `AMin..AMax`. It returns `False` if `Value` lies outside the specified range.

See also: `EnsureRange` ([614](#))

intpower

Synopsis: Return integer power.

Declaration: `function intpower(base: float;const exponent: Integer) : float`

Visibility: default

Description: `Intpower` returns `base` to the power `exponent`, where `exponent` is an integer value.

Errors: If `base` is zero and the `exponent` is negative, then an overflow error will occur.

See also: `power` ([631](#))

Listing: `./mathex/ex18.pp`

```

Program Example18;

{ Program to demonstrate the intpower function. }

Uses math;

Procedure DoIntpower (X : extended; Pow : Integer);

begin
  writeln(X:8:4, '^',Pow:2,' = ',intpower(X,pow):8:4);
end;

begin
  dointpower(0.0,0);
  dointpower(1.0,0);

```

```

dointpower(2.0,5);
dointpower(4.0,3);
dointpower(2.0,-1);
dointpower(2.0,-2);
dointpower(-2.0,4);
dointpower(-4.0,3);
end.

```

IsInfinite

Synopsis: Check whether value is infinite

Declaration: `function IsInfinite(const d: Double) : Boolean`

Visibility: default

Description: `IsInfinite` returns `True` if the double `d` contains the infinite value.

See also: `IsZero` ([619](#)), `IsInfinite` ([619](#))

IsNan

Synopsis: Check whether value is Not a Number

Declaration: `function IsNan(const d: Double) : Boolean`

Visibility: default

Description: `IsNan` returns `True` if the double `d` contains Not A Number (a value which cannot be represented correctly in double format).

See also: `IsZero` ([619](#)), `IsInfinite` ([619](#))

IsZero

Synopsis: Check whether value is zero

Declaration: `function IsZero(const A: Single;Epsilon: Single) : Boolean`
`function IsZero(const A: Single) : Boolean`
`function IsZero(const A: Extended;Epsilon: Extended) : Boolean`
`function IsZero(const A: Extended) : Boolean`

Visibility: default

Description: `IsZero` checks whether the float value `A` is zero, up to a precision of `Epsilon`. It returns `True` if `Abs(A)` is less than `Epsilon`.

The default value for `Epsilon` is dependent on the type of the arguments, but is `MinFloat` ([604](#)) for the float type.

See also: `IsNan` ([619](#)), `IsInfinite` ([619](#)), `SameValue` ([634](#))

ldexp

Synopsis: Return (2 to the power p) times x

Declaration: `function ldexp(x: float; const p: Integer) : float`

Visibility: default

Description: `Ldexp` returns (2 to the power p) times x.

Errors: None.

See also: `lnxp1` ([620](#)), `log10` ([621](#)), `log2` ([621](#)), `logn` ([622](#))

Listing: `./mathex/ex19.pp`

Program `Example19;`

{ Program to demonstrate the ldexp function. }

Uses `math;`

begin

`writeln(ldexp(2,4):8:4);`

`writeln(ldexp(0.5,3):8:4);`

end.

lnxp1

Synopsis: Return natural logarithm of 1+X

Declaration: `function lnxp1(x: float) : float`

Visibility: default

Description: `lnxp1` returns the natural logarithm of 1+X. The result is more precise for small values of x. x should be larger than -1.

Errors: If $x \leq -1$ then an `EInvalidArgument` exception will be raised.

See also: `ldexp` ([620](#)), `log10` ([621](#)), `log2` ([621](#)), `logn` ([622](#))

Listing: `./mathex/ex20.pp`

Program `Example20;`

{ Program to demonstrate the lnxp1 function. }

Uses `math;`

begin

`writeln(lnxp1(0));`

`writeln(lnxp1(0.5));`

`writeln(lnxp1(1));`

end.

log10

Synopsis: Return 10-Based logarithm.

Declaration: `function log10(x: float) : float`

Visibility: default

Description: Log10 returns the 10-base logarithm of X.

Errors: If x is less than or equal to 0 an 'invalid fpu operation' error will occur.

See also: `ldexp` (620), `lnxp1` (620), `log2` (621), `logn` (622)

Listing: `./mathex/ex21.pp`

Program Example21;

{ Program to demonstrate the log10 function. }

Uses math;

begin

```
  WriteLn(Log10(10):8:4);
  WriteLn(Log10(100):8:4);
  WriteLn(Log10(1000):8:4);
  WriteLn(Log10(1):8:4);
  WriteLn(Log10(0.1):8:4);
  WriteLn(Log10(0.01):8:4);
  WriteLn(Log10(0.001):8:4);
```

end.

log2

Synopsis: Return 2-based logarithm

Declaration: `function log2(x: float) : float`

Visibility: default

Description: Log2 returns the 2-base logarithm of X.

Errors: If x is less than or equal to 0 an 'invalid fpu operation' error will occur.

See also: `ldexp` (620), `lnxp1` (620), `log10` (621), `logn` (622)

Listing: `./mathex/ex22.pp`

Program Example22;

{ Program to demonstrate the log2 function. }

Uses math;

begin

```
  WriteLn(Log2(2):8:4);
  WriteLn(Log2(4):8:4);
  WriteLn(Log2(8):8:4);
  WriteLn(Log2(1):8:4);
```

```

WriteLn (Log2 (0.5):8:4);
WriteLn (Log2 (0.25):8:4);
WriteLn (Log2 (0.125):8:4);
end.

```

logn

Synopsis: Return N-based logarithm.

Declaration: `function logn(n: float;x: float) : float`

Visibility: default

Description: Logn returns the n-base logarithm of X.

Errors: If x is less than or equal to 0 an 'invalid fpu operation' error will occur.

See also: `ldexp` ([620](#)), `lnxp1` ([620](#)), `log10` ([621](#)), `log2` ([621](#))

Listing: `./mathex/ex23.pp`

Program Example23;

{ Program to demonstrate the logn function. }

Uses math;

begin

```

WriteLn (Logn (3,4):8:4);
WriteLn (Logn (2,4):8:4);
WriteLn (Logn (6,9):8:4);
WriteLn (Logn (exp (1),exp (1)):8:4);
WriteLn (Logn (0.5,1):8:4);
WriteLn (Logn (0.25,3):8:4);
WriteLn (Logn (0.125,5):8:4);
end.

```

Max

Synopsis: Return largest of 2 values

Declaration: `function Max(a: Integer;b: Integer) : Integer`
`function Max(a: Cardinal;b: Cardinal) : Cardinal`
`function Max(a: Int64;b: Int64) : Int64`
`function Max(a: Extended;b: Extended) : Extended`

Visibility: default

Description: Max returns the maximum of Int1 and Int2.

Errors: None.

See also: `min` ([626](#)), `maxIntValue` ([623](#)), `maxvalue` ([624](#))

Listing: `./mathex/ex24.pp`

```

Program Example24;

{ Program to demonstrate the max function. }

Uses math;

Var
  A,B : Cardinal;

begin
  A:=1;b:=2;
  writeln(max(a,b));
end.

```

MaxIntValue

Synopsis: Return largest element in integer array

Declaration: `function MaxIntValue(const Data: Array[] of Integer) : Integer`

Visibility: default

Description: MaxIntValue returns the largest integer out of the Data array.

This function is provided for Delphi compatibility, use the maxvalue ([624](#)) function instead.

Errors: None.

See also: maxvalue ([624](#)), minvalue ([627](#)), minIntValue ([627](#))

Listing: ./mathex/ex25.pp

```

Program Example25;

{ Program to demonstrate the MaxIntValue function. }

{ Make sure integer is 32 bit }
{$mode objfpc}

Uses math;

Type
  TExArray = Array[1..100] of Integer;

Var
  I : Integer;
  ExArray : TExArray;

begin
  Randomize;
  for I:=1 to 100 do
    ExArray[I]:=Random(I)-Random(100);
  Writeln(MaxIntValue(ExArray));
end.

```

maxvalue

Synopsis: Return largest value in array

Declaration: `function maxvalue(const data: Array[] of float) : float`
`function maxvalue(const data: Array[] of Integer) : Integer`
`function maxvalue(const data: PFloat;const N: Integer) : float`
`function maxvalue(const data: PInteger;const N: Integer) : Integer`

Visibility: default

Description: Maxvalue returns the largest value in the data array with integer or float values. The return value has the same type as the elements of the array.

The third and fourth forms accept a pointer to an array of N integer or float values.

Errors: None.

See also: `maxIntValue` (623), `minvalue` (627), `minIntValue` (627)

Listing: `./mathex/ex26.pp`

Program Example26;

{ Program to demonstrate the MaxValue function. }

{ Make sure integer is 32 bit }
{ \$mode objfpc }

Uses math;

Type

`TExFloatArray = Array[1..100] of Float;`
`TExIntArray = Array[1..100] of Integer;`

Var

`I : Integer;`
`ExFloatArray : TExFloatArray;`
`ExIntArray : TExIntArray;`
`AFloatArray : PFloat;`
`AlntArray : PInteger;`

begin

`Randomize;`
`AFloatArray := @ExFloatArray[1];`
`AlntArray := @ExIntArray[1];`
`for I:=1 to 100 do`
`ExFloatArray[I] := (Random-Random)*100;`
`for I:=1 to 100 do`
`ExIntArray[I] := Random(I)-Random(100);`
`Writeln('Max Float : ',MaxValue(ExFloatArray):8:4);`
`Writeln('Max Float (b) : ',MaxValue(AFloatArray,100):8:4);`
`Writeln('Max Integer : ',MaxValue(ExIntArray):8);`
`Writeln('Max Integer (b) : ',MaxValue(AlntArray,100):8);`

`end.`

mean

Synopsis: Return mean value of array

Declaration: `function mean(const data: Array[] of float) : float`
`function mean(const data: PFloat;const N: LongInt) : float`

Visibility: default

Description: Mean returns the average value of data. The second form accepts a pointer to an array of N values.

Errors: None.

See also: [meanandstddev \(625\)](#), [momentskewkurtosis \(628\)](#), [sum \(637\)](#)

Listing: ./mathex/ex27.pp

Program Example27;

{ Program to demonstrate the Mean function. }

Uses math;

Type

`TExArray = Array[1..100] of Float;`

Var

`I : Integer;`
`ExArray : TExArray;`

begin

`Randomize;`

`for I:=1 to 100 do`

`ExArray[I]:= (Random-Random)*100;`

`WriteLn('Max : ',MaxValue(ExArray):8:4);`

`WriteLn('Min : ',MinValue(ExArray):8:4);`

`WriteLn('Mean : ',Mean(ExArray):8:4);`

`WriteLn('Mean (b) : ',Mean(@ExArray[1],100):8:4);`

end.

meanandstddev

Synopsis: Return mean and standard deviation of array

Declaration: `procedure meanandstddev(const data: Array[] of float;var mean: float;`
`var stddev: float)`
`procedure meanandstddev(const data: PFloat;const N: LongInt;`
`var mean: float;var stddev: float)`

Visibility: default

Description: meanandstddev calculates the mean and standard deviation of data and returns the result in mean and stddev, respectively. Stddev is zero if there is only one value. The second form accepts a pointer to an array of N values.

Errors: None.

See also: [mean \(624\)](#), [sum \(637\)](#), [sumofsquares \(638\)](#), [momentskewkurtosis \(628\)](#)

Listing: ./mathex/ex28.pp

```

Program Example28;

{ Program to demonstrate the Meanandstddev function. }

Uses math;

Type
  TExArray = Array[1..100] of Extended;

Var
  I : Integer;
  ExArray : TExArray;
  Mean, stddev : Extended;

begin
  Randomize;
  for I:=1 to 100 do
    ExArray[I] := (Random-Random)*100;
  MeanAndStdDev( ExArray , Mean, StdDev );
  WriteLn( 'Mean      : ', Mean:8:4);
  WriteLn( 'StdDev    : ', StdDev:8:4);
  MeanAndStdDev( @ExArray[1], 100, Mean, StdDev );
  WriteLn( 'Mean      (b) : ', Mean:8:4);
  WriteLn( 'StdDev    (b) : ', StdDev:8:4);
end.

```

Min

Synopsis: Return smallest of two values.

Declaration: function Min(a: Integer;b: Integer) : Integer
 function Min(a: Cardinal;b: Cardinal) : Cardinal
 function Min(a: Int64;b: Int64) : Int64
 function Min(a: Extended;b: Extended) : Extended

Visibility: default

Description: min returns the smallest value of Int1 and Int2;

Errors: None.

See also: max ([622](#))

Listing: ./mathex/ex29.pp

```

Program Example29;

{ Program to demonstrate the min function. }

Uses math;

Var
  A,B : Cardinal;

begin
  A:=1;b:=2;
  writeln( min(a,b));
end.

```

MinIntValue

Synopsis: Return smallest value in integer array

Declaration: `function MinIntValue(const Data: Array[] of Integer) : Integer`

Visibility: default

Description: `MinIntValue` returns the smallest value in the `Data` array.

This function is provided for Delphi compatibility, use `minvalue` instead.

Errors: None

See also: `minvalue` ([627](#)), `maxIntValue` ([623](#)), `maxvalue` ([624](#))

Listing: `./mathex/ex30.pp`

Program `Example30;`

{ Program to demonstrate the MinIntValue function. }

{ Make sure integer is 32 bit }
{ \$mode objfpc }

Uses `math;`

Type

`TExArray = Array[1..100] of Integer;`

Var

`I : Integer;`
`ExArray : TExArray;`

begin

`Randomize;`

`for I:=1 to 100 do`

`ExArray[I]:=Random(I)-Random(100);`

`WriteLn(MinIntValue(ExArray));`

end.

minvalue

Synopsis: Return smallest value in array

Declaration: `function minvalue(const data: Array[] of float) : float`

`function minvalue(const data: Array[] of Integer) : Integer`

`function minvalue(const data: PFloat;const N: Integer) : float`

`function MinValue(const Data: PInteger;const N: Integer) : Integer`

Visibility: default

Description: `Minvalue` returns the smallest value in the `data` array with integer or float values. The return value has the same type as the elements of the array.

The third and fourth forms accept a pointer to an array of `N` integer or float values.

Errors: None.

See also: `maxIntValue` ([623](#)), `maxvalue` ([624](#)), `minIntValue` ([627](#))

Listing: ./mathex/ex31.pp

Program Example26;

```
{ Program to demonstrate the MinValue function. }
```

```
{ Make sure integer is 32 bit }
{$mode objfpc}
```

Uses math;

Type

```
TExFloatArray = Array[1..100] of Float;
TExIntArray = Array[1..100] of Integer;
```

Var

```
I : Integer;
ExFloatArray : TExFloatArray;
AFloatArray : PFloat;
ExIntArray : TExIntArray;
AIntArray : PInteger;
```

begin

```
  Randomize;
  AFloatArray := @ExFloatArray[0];
  AIntArray := @ExIntArray[0];
  for I:=1 to 100 do
    ExFloatArray[I] := (Random-Random)*100;
  for I:=1 to 100 do
    ExIntArray[I] := Random(I)-Random(100);
  Writeln('Min Float      : ', MinValue(ExFloatArray):8:4);
  Writeln('Min Float    (b) : ', MinValue(AFloatArray,100):8:4);
  Writeln('Min Integer   : ', MinValue(ExIntArray):8);
  Writeln('Min Integer (b) : ', MinValue(AIntArray,100):8);
```

end.

momentskewkurtosis

Synopsis: Return 4 first moments of distribution

Declaration:

```
procedure momentskewkurtosis(const data: Array[] of float; var m1: float;
                             var m2: float; var m3: float; var m4: float;
                             var skew: float; var kurtosis: float)
procedure momentskewkurtosis(const data: PFloat; const N: Integer;
                             var m1: float; var m2: float; var m3: float;
                             var m4: float; var skew: float;
                             var kurtosis: float)
```

Visibility: default

Description: momentskewkurtosis calculates the 4 first moments of the distribution of values in data and returns them in m1,m2,m3 and m4, as well as the skew and kurtosis.

Errors: None.

See also: mean ([624](#)), meanandstddev ([625](#))

Listing: ./mathex/ex32.pp

Program Example32;

{ Program to demonstrate the momentskewkurtosis function. }

Uses math;

Var

DistArray : **Array**[1..1000] **of** float;
 I : longint;
 m1,m2,m3,m4,skew,kurtosis : float;

begin

randomize;
 for I:=1 **to** 1000 **do**
 distarray[I]:=random;
 momentskewkurtosis(DistArray,m1,m2,m3,m4,skew,kurtosis);

Writeln ('1st moment : ',m1:8:6);
 Writeln ('2nd moment : ',m2:8:6);
 Writeln ('3rd moment : ',m3:8:6);
 Writeln ('4th moment : ',m4:8:6);
 Writeln ('Skew : ',skew:8:6);
 Writeln ('kurtosis : ',kurtosis:8:6);

end.

norm

Synopsis: Return Euclidian norm

Declaration: function norm(const data: Array[] of float) : float
 function norm(const data: PFloat;const N: Integer) : float

Visibility: default

Description: Norm calculates the Euclidian norm of the array of data. This equals sqrt (sumofsquares (data)).

The second form accepts a pointer to an array of N values.

Errors: None.

See also: sumofsquares ([638](#))

Listing: ./mathex/ex33.pp

Program Example33;

{ Program to demonstrate the norm function. }

Uses math;

Type

TVector = **Array**[1..10] **of** Float;

Var

AVector : Tvector;
 I : longint;

begin

```

for i:=1 to 10 do
  Avector[i]:=Random;
  WriteIn(Norm(AVector));
end.

```

popnstddev

Synopsis: Return population variance

Declaration: `function popnstddev(const data: Array[] of float) : float`
`function popnstddev(const data: PFloat;const N: Integer) : float`

Visibility: default

Description: Popnstddev returns the square root of the population variance of the values in the Data array. It returns zero if there is only one value.

The second form of this function accepts a pointer to an array of N values.

Errors: None.

See also: popnvariance (630), mean (624), meanandstddev (625), stddev (637), momentskewkurtosis (628)

Listing: ./mathex/ex35.pp

Program Example35;

{ Program to demonstrate the PopnStdDev function. }

Uses Math;

Type

TExArray = **Array**[1..100] **of** Float;

Var

i : Integer;
ExArray : TExArray;

begin

Randomize;

for i:=1 **to** 100 **do**

ExArray[i]:=(Random-~~Random~~)*100;

WriteIn('Max : ',MaxValue(ExArray):8:4);

WriteIn('Min : ',MinValue(ExArray):8:4);

WriteIn('Pop. stddev. : ',PopnStdDev(ExArray):8:4);

WriteIn('Pop. stddev. (b) : ',PopnStdDev(@ExArray[1],100):8:4);

end.

popnvariance

Synopsis: Return population variance

Declaration: `function popnvariance(const data: PFloat;const N: Integer) : float`
`function popnvariance(const data: Array[] of float) : float`

Visibility: default

Description: `Popnvariance` returns the square root of the population variance of the values in the `Data` array. It returns zero if there is only one value.

The second form of this function accepts a pointer to an array of `N` values.

Errors: None.

See also: `popnstddev` (630), `mean` (624), `meanandstddev` (625), `stddev` (637), `momentskewkurtosis` (628)

Listing: `./mathex/ex36.pp`

Program `Example36`;

{ Program to demonstrate the PopnVariance function. }

Uses `math`;

Type

`TExArray = Array[1..100] of Float`;

Var

`I : Integer`;

`ExArray : TExArray`;

begin

`Randomize`;

for `I:=1 to 100 do`

`ExArray[I]:= (Random-Random)*100`;

`WriteIn` ('Max : ', `MaxValue(ExArray):8:4`);

`WriteIn` ('Min : ', `MinValue(ExArray):8:4`);

`WriteIn` ('Pop. var. : ', `PopnVariance(ExArray):8:4`);

`WriteIn` ('Pop. var. (b) : ', `PopnVariance(@ExArray[1],100):8:4`);

end.

power

Synopsis: Return real power.

Declaration: `function power(base: float;exponent: float) : float`

Visibility: `default`

Description: `power` raises `base` to the power `power`. This is equivalent to `exp(power*ln(base))`. Therefore `base` should be non-negative.

Errors: None.

See also: `intpower` (618)

Listing: `./mathex/ex34.pp`

Program `Example34`;

{ Program to demonstrate the power function. }

Uses `Math`;

procedure `dopower(x,y : float)`;

```

begin
  writeln(x:8:6, '^', y:8:6, ' = ', power(x,y):8:6)
end;

begin
  dopower(2,2);
  dopower(2,-2);
  dopower(2,0.0);
end.

```

radtcycle

Synopsis: Convert radians angle to cycle angle

Declaration: `function radtcycle(rad: float) : float`

Visibility: default

Description: Radtcycle converts its argument rad (an angle expressed in radians) to an angle in cycles.
(1 cycle equals 2 pi radians)

Errors: None.

See also: degtograd ([613](#)), degtorad ([613](#)), radtodeg ([632](#)), radtograd ([633](#)), cycletorad ([612](#))

Listing: ./mathex/ex37.pp

Program Example37;

{ Program to demonstrate the radtcycle function. }

Uses math;

```

begin
  writeln(radtcycle(2*pi):8:6);
  writeln(radtcycle(pi):8:6);
  writeln(radtcycle(pi/2):8:6);
end.

```

radtodeg

Synopsis: Convert radians angle to degrees angle

Declaration: `function radtodeg(rad: float) : float`

Visibility: default

Description: Radtodeg converts its argument rad (an angle expressed in radians) to an angle in degrees. (180 degrees equals pi radians)

Errors: None.

See also: degtograd ([613](#)), degtorad ([613](#)), radtodeg ([632](#)), radtograd ([633](#)), cycletorad ([612](#))

Listing: ./mathex/ex38.pp

Program Example38;

{ Program to demonstrate the radtodeg function. }

Uses math;

begin

writeln (radtodeg(2***pi**):8:6);

writeln (radtodeg(**pi**):8:6);

writeln (radtodeg(**pi**/2):8:6);

end.

radtograd

Synopsis: Convert radians angle to grads angle

Declaration: function radtograd(rad: float) : float

Visibility: default

Description: Radtodeg converts its argument rad (an angle expressed in radians) to an angle in grads.
(200 grads equals pi radians)

Errors: None.

See also: degtograd ([613](#)), degtorad ([613](#)), radto cycle ([632](#)), radtodeg ([632](#)), cycletorad ([612](#))

Listing: ./mathex/ex39.pp

Program Example39;

{ Program to demonstrate the radtograd function. }

Uses math;

begin

writeln (radtograd(2***pi**):8:6);

writeln (radtograd(**pi**):8:6);

writeln (radtograd(**pi**/2):8:6);

end.

randg

Synopsis: Return gaussian distributed random number.

Declaration: function randg(mean: float; stddev: float) : float

Visibility: default

Description: randg returns a random number which - when produced in large quantities - has a Gaussian distribution with mean mean and standarddeviation stddev.

Errors: None.

See also: mean ([624](#)), stddev ([637](#)), meanandstddev ([625](#))

Listing: ./mathex/ex40.pp

Program Example40;

{ Program to demonstrate the randg function. }

Uses Math;

Type

TExArray = **Array**[1..10000] **of** Float;

Var

I : Integer;

ExArray : TExArray;

Mean, stddev : Float;

begin

Randomize;

for I:=1 **to** 10000 **do**

ExArray[I]:=Randg(1,0.2);

MeanAndStdDev(ExArray,Mean,StdDev);

WriteLn('Mean : ',Mean:8:4);

WriteLn('StdDev : ',StdDev:8:4);

end.

SameValue

Synopsis: Check whether 2 float values are the same

Declaration: function SameValue(const A: Extended;const B: Extended) : Boolean
 function SameValue(const A: Single;const B: Single) : Boolean
 function SameValue(const A: Extended;const B: Extended;
 Epsilon: Extended) : Boolean
 function SameValue(const A: Single;const B: Single;Epsilon: Single)
 : Boolean

Visibility: default

Description: SameValue returns True if the floating-point values A and B are the same, i.e. whether the absolute value of their difference is smaller than Epsilon. If their difference is larger, then False is returned.

The default value for Epsilon is dependent on the type of the arguments, but is MinFloat (604) for the float type.

See also: MinFloat (604), IsZero (619)

SetExceptionMask

Synopsis: Set the Floating Point Unit exception mask.

Declaration: function SetExceptionMask(const Mask: TFPUEExceptionMask)
 : TFPUEExceptionMask

Visibility: default

Description: Set the Floating Point Unit exception mask.

SetPrecisionMode

Synopsis: Set the Floating Point Unit precision mode.

Declaration: `function SetPrecisionMode(const Precision: TFPUPrecisionMode)
: TFPUPrecisionMode`

Visibility: default

Description: Set the Floating Point Unit precision mode.

SetRoundMode

Synopsis: Set the Floating Point Unit rounding mode.

Declaration: `function SetRoundMode(const RoundMode: TFPURoundingMode)
: TFPURoundingMode`

Visibility: default

Description: Set the Floating Point Unit rounding mode.

SetSSECSR

Declaration: `procedure SetSSECSR(w: dword)`

Visibility: default

Sign

Synopsis: Return sign of argument

Declaration: `function Sign(const AValue: Integer) : TValueSign
function Sign(const AValue: Int64) : TValueSign
function Sign(const AValue: Double) : TValueSign`

Visibility: default

Description: `Sign` returns the sign of its argument, which can be an Integer, 64 bit integer, or a double. The returned value is an integer which is -1, 0 or 1, and can be used to do further calculations with.

sincos

Synopsis: Return sine and cosine of argument

Declaration: `procedure sincos(theta: float; var sinus: float; var cosinus: float)`

Visibility: default

Description: `Sincos` calculates the sine and cosine of the angle `theta`, and returns the result in `sinus` and `cosinus`.

On Intel hardware, This calculation will be faster than making 2 calls to calculate the sine and cosine separately.

Errors: None.

See also: [arcsin \(608\)](#), [arccos \(606\)](#)

Listing: ./mathex/ex41.pp

Program Example41;

{ Program to demonstrate the sincos function. }

Uses math;

Procedure dosincos(Angle : Float);

Var

Sine, Cosine : Float;

begin

sincos(angle, sine, cosine);

Write('Angle : ', Angle:8:6);

Write(' Sine : ', sine:8:6);

Write(' Cosine : ', cosine:8:6);

end;

begin

dosincos(**pi**);

dosincos(**pi**/2);

dosincos(**pi**/3);

dosincos(**pi**/4);

dosincos(**pi**/6);

end.

sinh

Synopsis: Return hyperbolic sine

Declaration: function sinh(x: float) : float

Visibility: default

Description: Sinh returns the hyperbolic sine of its argument x.

Errors:

See also: cosh ([611](#)), arsinh ([610](#)), tanh ([640](#)), artanh ([610](#))

Listing: ./mathex/ex42.pp

Program Example42;

{ Program to demonstrate the sinh function. }

Uses math;

begin

writeln(sinh(0));

writeln(sinh(1));

writeln(sinh(-1));

end.

stddev

Synopsis: Return standard deviation of data

Declaration: `function stddev(const data: Array[] of float) : float`
`function stddev(const data: PFloat;const N: Integer) : float`

Visibility: default

Description: Stddev returns the standard deviation of the values in Data. It returns zero if there is only one value.

The second form of the function accepts a pointer to an array of N values.

Errors: None.

See also: mean ([624](#)), meanandstddev ([625](#)), variance ([641](#)), totalvariance ([641](#))

Listing: ./mathex/ex43.pp

Program Example40;

{ Program to demonstrate the stddev function . }

Uses Math;

Type

TExArray = **Array**[1..10000] of Float;

Var

I : Integer;

ExArray : TExArray;

begin

Randomize;

for I:=1 **to** 10000 **do**

ExArray[I]:=Randg(1,0.2);

WriteIn('StdDev : ',StdDev(ExArray):8:4);

WriteIn('StdDev (b) : ',StdDev(@ExArray[0],10000):8:4);

end.

sum

Synopsis: Return sum of values

Declaration: `function sum(const data: Array[] of float) : float`
`function sum(const data: PFloat;const N: LongInt) : float`

Visibility: default

Description: Sum returns the sum of the values in the data array.

The second form of the function accepts a pointer to an array of N values.

Errors: None.

See also: sumofsquares ([638](#)), sumsandsquares ([639](#)), totalvariance ([641](#)), variance ([641](#))

Listing: ./mathex/ex44.pp

```

Program Example44;

{ Program to demonstrate the Sum function. }

Uses math;

Type
  TExArray = Array[1..100] of Float;

Var
  I : Integer;
  ExArray : TExArray;

begin
  Randomize;
  for I:=1 to 100 do
    ExArray[I]:=(RandomRandom)*100;
  WriteIn( 'Max      : ',MaxValue(ExArray):8:4);
  WriteIn( 'Min      : ',MinValue(ExArray):8:4);
  WriteIn( 'Sum      : ',Sum(ExArray):8:4);
  WriteIn( 'Sum (b) : ',Sum(@ExArray[1],100):8:4);
end.

```

sumofsquares

Synopsis: Return sum of squares of values

Declaration: `function sumofsquares(const data: Array[] of float) : float`
`function sumofsquares(const data: PFloat;const N: Integer) : float`

Visibility: default

Description: Sumofsquares returns the sum of the squares of the values in the data array.

The second form of the function accepts a pointer to an array of N values.

Errors: None.

See also: [sum \(637\)](#), [sumsandsquares \(639\)](#), [totalvariance \(641\)](#), [variance \(641\)](#)

Listing: ./mathex/ex45.pp

```

Program Example45;

{ Program to demonstrate the SumOfSquares function. }

Uses math;

Type
  TExArray = Array[1..100] of Float;

Var
  I : Integer;
  ExArray : TExArray;

begin
  Randomize;
  for I:=1 to 100 do

```

```

    ExArray[i] := (Random-Random)*100;
    Writeln('Max           : ',MaxValue(ExArray):8:4);
    Writeln('Min           : ',MinValue(ExArray):8:4);
    Writeln('Sum squares   : ',SumOfSquares(ExArray):8:4);
    Writeln('Sum squares (b) : ',SumOfSquares(@ExArray[1],100):8:4);
end.

```

sumsandsquares

Synopsis: Return sum and sum of squares of values.

Declaration: `procedure sumsandsquares(const data: Array[] of float; var sum: float; var sumofsquares: float)`
`procedure sumsandsquares(const data: PFloat; const N: Integer; var sum: float; var sumofsquares: float)`

Visibility: default

Description: `sumsandsquares` calculates the sum of the values and the sum of the squares of the values in the `data` array and returns the results in `sum` and `sumofsquares`.

The second form of the function accepts a pointer to an array of `N` values.

Errors: None.

See also: `sum` (637), `sumofsquares` (638), `totalvariance` (641), `variance` (641)

Listing: `./mathex/ex46.pp`

Program Example45;

{ Program to demonstrate the SumOfSquares function. }

Uses math;

Type

`TExArray = Array[1..100] of Float;`

Var

`I : Integer;`
`ExArray : TExArray;`
`s,ss : float;`

begin

```

    Randomize;
    for I:=1 to 100 do
        ExArray[i] := (Random-Random)*100;
    Writeln('Max           : ',MaxValue(ExArray):8:4);
    Writeln('Min           : ',MinValue(ExArray):8:4);
    SumsAndSquares(ExArray,S,SS);
    Writeln('Sum           : ',S:8:4);
    Writeln('Sum squares   : ',SS:8:4);
    SumsAndSquares(@ExArray[1],100,S,SS);
    Writeln('Sum (b)       : ',S:8:4);
    Writeln('Sum squares (b) : ',SS:8:4);

```

end.

tan

Synopsis: Return tangent

Declaration: `function tan(x: float) : float`

Visibility: default

Description: Tan returns the tangent of x .

Errors: If x (normalized) is $\pi/2$ or $3\pi/2$ then an overflow will occur.

See also: `tanh` (640), `arcsin` (608), `sincos` (635), `arccos` (606)

Listing: `./mathex/ex47.pp`

Program Example47;

{ Program to demonstrate the Tan function. }

Uses math;

Procedure DoTan(Angle : Float);

begin

Write('Angle : ', RadToDeg(Angle):8:6);

WriteLn(' Tangent : ', Tan(Angle):8:6);

end;

begin

 DoTan(0);

 DoTan(**Pi**);

 DoTan(**Pi**/3);

 DoTan(**Pi**/4);

 DoTan(**Pi**/6);

end.

tanh

Synopsis: Return hyperbolic tangent

Declaration: `function tanh(x: float) : float`

Visibility: default

Description: Tanh returns the hyperbolic tangent of x .

Errors: None.

See also: `arcsin` (608), `sincos` (635), `arccos` (606)

Listing: `./mathex/ex48.pp`

Program Example48;

{ Program to demonstrate the Tanh function. }

Uses math;

```

begin
  writeln(tanh(0));
  writeln(tanh(1));
  writeln(tanh(-1));
end.

```

totalvariance

Synopsis: Return total variance of values

Declaration: `function totalvariance(const data: Array[] of float) : float`
`function totalvariance(const data: PFloat; const N: Integer) : float`

Visibility: default

Description: `TotalVariance` returns the total variance of the values in the data array. It returns zero if there is only one value.

The second form of the function accepts a pointer to an array of N values.

Errors: None.

See also: `variance` ([641](#)), `stddev` ([637](#)), `mean` ([624](#))

Listing: ./mathex/ex49.pp

Program Example49;

{ Program to demonstrate the TotalVariance function. }

Uses math;

Type

`TExArray = Array[1..100] of Float;`

Var

`I : Integer;`
`ExArray : TExArray;`
`TV : float;`

begin

`Randomize;`
`for I:=1 to 100 do`
`ExArray[I]:=(Random-Random)*100;`
`TV:=TotalVariance(ExArray);`
`writeln('Total variance : ',TV:8:4);`
`TV:=TotalVariance(@ExArray[1],100);`
`writeln('Total Variance (b) : ',TV:8:4);`
end.

variance

Synopsis: Return variance of values

Declaration: `function variance(const data: Array[] of float) : float`
`function variance(const data: PFloat; const N: Integer) : float`

Visibility: default

Description: `Variance` returns the variance of the values in the `data` array. It returns zero if there is only one value.

The second form of the function accepts a pointer to an array of `N` values.

Errors: None.

See also: `totalvariance` (641), `stddev` (637), `mean` (624)

Listing: `./mathex/ex50.pp`

Program `Example50`;

{ Program to demonstrate the Variance function. }

Uses `math`;

Type

`TExArray = Array[1..100] of Float`;

Var

`I : Integer`;
`ExArray : TExArray`;
`V : float`;

begin

`Randomize`;
for `I:=1 to 100 do`
 `ExArray[I]:= (Random-Random)*100`;
`V:=Variance(ExArray)`;
 WriteLn('Variance : ',V:8:4);
`V:=Variance(@ExArray[1],100)`;
 WriteLn('Variance (b) : ',V:8:4);

end.

17.13 `EINVAL` argument

Description

Exception raised when invalid arguments are passed to a function.

Chapter 18

Reference for unit 'mmx'

18.1 Overview

This document describes the MMX unit. This unit allows you to use the MMX capabilities of the Free Pascal compiler. It was written by Florian Klaempfl for the I386 processor. It should work on all platforms that use the Intel processor.

18.2 Constants, types and variables

Constants

```
is_amd_3d_cpu : Boolean = false
```

The `is_amd_3d_cpu` initialized constant allows you to determine if the computer has the AMD 3D extensions. It is set correctly in the unit's initialization code.

```
is_amd_3d_dsp_cpu : Boolean = false
```

The `is_amd_3d_dsp_cpu` initialized constant allows you to determine if the computer has the AMD 3D DSP extensions. It is set correctly in the unit's initialization code.

```
is_amd_3d_mmx_cpu : Boolean = false
```

The `is_amd_3d_mmx_cpu` initialized constant allows you to determine if the computer has the AMD 3D MMX extensions. It is set correctly in the unit's initialization code.

```
is_mmx_cpu : Boolean = false
```

The `is_mmx_cpu` initialized constant allows you to determine if the computer has MMX extensions. It is set correctly in the unit's initialization code.

```
is_sse2_cpu : Boolean = false
```

The `is_sse2_cpu` initialized constant allows you to determine if the computer has the SSE2 extensions. It is set correctly in the unit's initialization code.

```
is_sse_cpu : Boolean = false
```

The `is_sse_cpu` initialized constant allows you to determine if the computer has the SSE extensions. It is set correctly in the unit's initialization code.

Types

`pmmxbyte = ^tmmxbyte`

Pointer to `tmmxbyte` (644) array type

`pmmxcardinal = ^tmmxcardinal`

Pointer to `tmmxcardinal` (644) array type

`pmmxinteger = ^tmmxinteger`

Pointer to `tmmxinteger` (644) array type

`pmmxlongint = ^tmmxlongint`

Pointer to `tmmxlongint` (644) array type

`pmmxshortint = ^tmmxshortint`

Pointer to `tmmxshortint` (644) array type

`pmmxsingle = ^tmmxsingle`

Pointer to `tmmxsingle` (644) array type

`pmmxword = ^tmmxword`

Pointer to `tmmxword` (644) array type

`tmmxbyte = Array[0..7] of Byte`

Array of bytes, 64 bits in size

`tmmxcardinal = Array[0..1] of cardinal`

Array of cardinals, 64 bits in size

`tmmxinteger = Array[0..3] of Integer`

Array of integers, 64 bits in size

`tmmxlongint = Array[0..1] of LongInt`

Array of longint, 64 bits in size

`tmmxshortint = Array[0..7] of ShortInt`

Array of shortints, 64 bits in size

`tmmxsingle = Array[0..1] of single`

Array of singles, 64 bits in size

`tmmxword = Array[0..3] of Word`

Array of words, 64 bits in size

18.3 Procedures and functions

emms

Synopsis: Reset floating point registers

Declaration: `procedure emms`

Visibility: `default`

Description: `Emms` sets all floating point registers to empty. This procedure must be called after you have used any MMX instructions, if you want to use floating point arithmetic. If you just want to move floating point data around, it isn't necessary to call this function, the compiler doesn't use the FPU registers when moving data. Only when doing calculations, you should use this function. The following code demonstrates this:

```
Program MMXDemo;
uses mmx;
var
  d1 : double;
  a : array[0..10000] of double;
  i : longint;
begin
  d1:=1.0;
  {$mmx+}
  { floating point data is used, but we do _no_ arithmetic }
  for i:=0 to 10000 do
    a[i]:=d2; { this is done with 64 bit moves }
  {$mmx-}
  emms; { clear fpu }
  { now we can do floating point arithmetic again }
end.
```

See also: `femms` ([645](#))

femms

Synopsis: Reset floating point registers - AMD version

Declaration: `procedure femms`

Visibility: `default`

Description: `femms` executes the `femms` assembler instruction for AMD processors. it is not supported by all assemblers, hence it is coded as byte codes.

See also: `emms` ([645](#))

Chapter 19

Reference for unit 'Mouse'

19.1 Writing a custom mouse driver

The `mouse` unit has support for adding a custom mouse driver. This can be used to add support for mice not supported by the standard Free Pascal driver, but also to enhance an existing driver for instance to log mouse events or to implement a record and playback function.

The following unit shows how a mouse driver can be enhanced by adding some logging capabilities to the driver.

19.2 Overview

The `Mouse` unit implements a platform independent mouse handling interface. It is implemented identically on all platforms supported by Free Pascal and can be enhanced with custom drivers, should this be needed. It is intended to be used only in text-based screens, for instance in conjunction with the keyboard and video unit. No support for graphical screens is implemented, and there are (currently) no plans to implement this.

19.3 Constants, types and variables

Constants

`errMouseBase = 1030`

Base for mouse error codes.

`errMouseInitError = errMouseBase + 0`

Mouse initialization error

`errMouseNotImplemented = errMouseBase + 1`

Mouse driver not implemented.

`MouseActionDown = $0001`

Mouse button down event signal.

MouseActionMove = \$0004

Mouse cursor move event signal.

MouseActionUp = \$0002

Mouse button up event signal.

MouseEventBufSize = 16

The mouse unit has a mechanism to buffer mouse events. This constant defines the size of the event buffer.

MouseLeftButton = \$01

Left mouse button event.

MouseMiddleButton = \$04

Middle mouse button event.

MouseRightButton = \$02

Right mouse button event.

Types

PMouseEvent = ^TMouseEvent

Pointer to TMouseEvent ([648](#)) record.

```
TMouseDriver = record
  UseDefaultQueue : Boolean;
  InitDriver : procedure;
  DoneDriver : procedure;
  DetectMouse : function : Byte;
  ShowMouse : procedure;
  HideMouse : procedure;
  GetMouseX : function : Word;
  GetMouseY : function : Word;
  GetMouseButtons : function : Word;
  SetMouseXY : procedure(x: Word;y: Word);
  GetMouseEvent : procedure(var MouseEvent: TMouseEvent);
  PollMouseEvent : function(var MouseEvent: TMouseEvent) : Boolean;
  PutMouseEvent : procedure(const MouseEvent: TMouseEvent);
end
```

The TMouseDriver record is used to implement a mouse driver in the SetMouseDriver ([653](#)) function. Its fields must be filled in before calling the SetMouseDriver ([653](#)) function.

```
TMouseEvent = packed record
  buttons : Word;
```

```

    x : Word;
    y : Word;
    Action : Word;
end

```

The `TMouseEvent` is the central type of the mouse unit, it is used to describe all mouse events.

The `Buttons` field describes which buttons were down when the event occurred. The `x,y` fields describe where the event occurred on the screen. The `Action` describes what action was going on when the event occurred. The `Buttons` and `Action` field can be examined using the constants defined in the unit interface.

Variables

`MouseButtons` : Byte

This variable keeps track of the last known mouse button state. Do not use.

`MouseIntFlag` : Byte

This variable keeps track of the last known internal mouse state. Do not use.

`MouseWhereX` : Word

This variable keeps track of the last known cursor position. Do not use.

`MouseWhereY` : Word

This variable keeps track of the last known cursor position. Do not use.

19.4 Procedures and functions

DetectMouse

Synopsis: Detect the presence of a mouse.

Declaration: `function DetectMouse` : Byte

Visibility: default

Description: `DetectMouse` detects whether a mouse is attached to the system or not. If there is no mouse, then zero is returned. If a mouse is attached, then the number of mouse buttons is returned.

This function should be called after the mouse driver was initialized.

Errors: None.

See also: `InitMouse` ([652](#)), `DoneMouse` ([649](#))

Listing: `./mouseex/ex1.pp`

Program `Example1`;

{ Program to demonstrate the DetectMouse function. }

Uses mouse;

Var

Buttons : Byte;

begin

InitMouse;

Buttons:=DetectMouse;

If Buttons=0 **then**

WriteLn('No mouse present. ')

else

WriteLn('Found mouse with ',Buttons, ' buttons. ');

DoneMouse;

end.

DoneMouse

Synopsis: Deinitialize mouse driver.

Declaration: procedure DoneMouse

Visibility: default

Description: DoneMouse De-initializes the mouse driver. It cleans up any memory allocated when the mouse was initialized, or removes possible mouse hooks from memory. The mouse functions will not work after DoneMouse was called. If DoneMouse is called a second time, it will exit at once. InitMouse should be called before DoneMouse can be called again.

For an example, see most other mouse functions.

Errors: None.

See also: DetectMouse ([648](#)), InitMouse ([652](#))

GetMouseButtons

Synopsis: Get the state of the mouse buttons

Declaration: function GetMouseButtons : Word

Visibility: default

Description: GetMouseButtons returns the current button state of the mouse, i.e. it returns a or-ed combination of the following constants:

MouseLeftButton When the left mouse button is held down.

MouseRightButton When the right mouse button is held down.

MouseMiddleButton When the middle mouse button is held down.

Errors: None.

See also: GetMouseEvent ([650](#)), GetMouseX ([650](#)), GetMouseY ([651](#))

Listing: ./mouseex/ex2.pp

Program Example2;

{ Program to demonstrate the GetMouseButtons function. }

Uses mouse;

begin

 InitMouse;

WriteLn('Press right mouse button to exit program');

While (GetMouseButtons<>MouseRightButton) **do** ;

 DoneMouse;

end.

GetMouseDriver

Synopsis: Get a copy of the currently active mouse driver.

Declaration: `procedure GetMouseDriver(var Driver: TMouseDriver)`

Visibility: default

Description: `GetMouseDriver` returns the currently set mouse driver. It can be used to retrieve the current mouse driver, and override certain callbacks.

A more detailed explanation about getting and setting mouse drivers can be found in [mousedrv \(646\)](#).

For an example, see the section on writing a custom mouse driver, [mousedrv \(646\)](#)

Errors: None.

See also: [SetMouseDriver \(653\)](#)

GetMouseEvent

Synopsis: Get next mouse event from the queue.

Declaration: `procedure GetMouseEvent(var MouseEvent: TMouseEvent)`

Visibility: default

Description: `GetMouseEvent` returns the next mouse event (a movement, button press or button release), and waits for one if none is available in the queue.

Some mouse drivers can implement a mouse event queue which can hold multiple events till they are fetched. Others don't, and in that case, a one-event queue is implemented for use with `PollMouseEvent (653)`.

Errors: None.

See also: [GetMouseButtons \(649\)](#), [GetMouseX \(650\)](#), [GetMouseY \(651\)](#)

GetMouseX

Synopsis: Query the current horizontal position of the mouse cursor.

Declaration: `function GetMouseX : Word`

Visibility: default

Description: `GetMouseX` returns the current X position of the mouse. X is measured in characters, starting at 0 for the left side of the screen.

Errors: None.

See also: `GetMouseButtons` ([649](#)), `GetMouseEvent` ([650](#)), `GetMouseY` ([651](#))

Listing: `./mouseex/ex4.pp`

Program `Example4`;

{ Program to demonstrate the GetMouseX,GetMouseY functions. }

Uses `mouse`;

Var

`X,Y : Word`;

begin

`InitMouse`;

WriteIn ('Move mouse cursor to square 10,10 to end');

Repeat

`X:=GetMouseX`;

`Y:=GetMouseY`;

WriteIn ('X,Y= (' ,X, ' , ' ,Y, ') ');

Until (X=9) **and** (Y=9);

`DoneMouse`;

end.

GetMouseY

Synopsis: Query the current vertical position of the mouse cursor.

Declaration: `function GetMouseY : Word`

Visibility: `default`

Description: `GetMouseY` returns the current Y position of the mouse. Y is measured in characters, starting at 0 for the top of the screen.

For an example, see `GetMouseX` ([650](#))

Errors: None.

See also: `GetMouseButtons` ([649](#)), `GetMouseEvent` ([650](#)), `GetMouseX` ([650](#))

HideMouse

Synopsis: Hide the mouse cursor.

Declaration: `procedure HideMouse`

Visibility: `default`

Description: `HideMouse` hides the mouse cursor. This may or may not be implemented on all systems, and depends on the driver.

Errors: None.

See also: ShowMouse ([654](#))

Listing: ./mouseex/ex5.pp

Program Example5;

{ Program to demonstrate the HideMouse function. }

Uses mouse;

Var

Event : TMouseEvent;

Visible : Boolean;

begin

InitMouse;

ShowMouse;

Visible:=True;

WriteLn('Press left mouse button to hide/show, right button quits');

Repeat

GetMouseEvent(Event);

With Event **do**

If (Buttons=MouseLeftbutton) **and**
(Action=MouseActionDown) **then**

begin

If Visible **then**

HideMouse

else

ShowMouse;

Visible:=**Not** Visible;

end;

Until (Event.Buttons=MouseRightButton) **and**
(Event.Action=MouseActionDown);

DoneMouse;

end.

InitMouse

Synopsis: Initialize the FPC mouse driver.

Declaration: procedure InitMouse

Visibility: default

Description: InitMouse Initializes the mouse driver. This will allocate any data structures needed for the mouse to function. All mouse functions can be used after a call to InitMouse.

A call to InitMouse must always be followed by a call to DoneMouse ([649](#)) at program exit. Failing to do so may leave the mouse in an unusable state, or may result in memory leaks.

For an example, see most other functions.

Errors: None.

See also: DoneMouse ([649](#)), DetectMouse ([648](#))

PollMouseEvent

Synopsis: Query next mouse event. Do not wait if none available.

Declaration: `function PollMouseEvent(var MouseEvent: TMouseEvent) : Boolean`

Visibility: default

Description: `PollMouseEvent` checks whether a mouse event is available, and returns it in `MouseEvent` if one is found. The function result is `True` in that case. If no mouse event is pending, the function result is `False`, and the contents of `MouseEvent` is undefined.

Note that after a call to `PollMouseEvent`, the event should still be removed from the mouse event queue with a call to `GetMouseEvent`.

Errors: None.

See also: `GetMouseEvent` ([650](#)), `PutMouseEvent` ([653](#))

PutMouseEvent

Synopsis: Put a mouse event in the venet queue.

Declaration: `procedure PutMouseEvent(const MouseEvent: TMouseEvent)`

Visibility: default

Description: `PutMouseEvent` adds `MouseEvent` to the input queue. The next call to `GetMouseEvent` ([650](#)) or `PollMouseEvent` will then return `MouseEvent`.

Please note that depending on the implementation the mouse event queue can hold only one value.

Errors: None.

See also: `GetMouseEvent` ([650](#)), `PollMouseEvent` ([653](#))

SetMouseDriver

Synopsis: Set a new mouse driver.

Declaration: `procedure SetMouseDriver(const Driver: TMouseDriver)`

Visibility: default

Description: `SetMouseDriver` sets the mouse driver to `Driver`. This function should be called before `InitMouse` ([652](#)) is called, or after `DoneMouse` is called. If it is called after the mouse has been initialized, it does nothing.

For more information on setting the mouse driver, `mousedrv` ([646](#)).

For an example, see `mousedrv` ([646](#))

Errors:

See also: `InitMouse` ([652](#)), `DoneMouse` ([649](#)), `GetMouseDriver` ([650](#))

SetMouseXY

Synopsis: Set the mouse cursor position.

Declaration: `procedure SetMouseXY(x: Word; y: Word)`

Visibility: default

Description: `SetMouseXY` places the mouse cursor on `X`, `Y`. `X` and `Y` are zero based character coordinates: 0, 0 is the top-left corner of the screen, and the position is in character cells (i.e. not in pixels).

Errors: None.

See also: `GetMouseX` ([650](#)), `GetMouseY` ([651](#))

Listing: `./mouseex/ex7.pp`

Program `Example7`;

{ Program to demonstrate the SetMouseXY function. }

Uses `mouse`;

begin

`InitMouse`;

`WriteLn`('Click right mouse button to quit.');

`SetMouseXY`(40,12);

Repeat

`WriteLn`(`GetMouseX`, ' ', `GetMouseY`);

If (`GetMouseX`>70) **then**

`SetMouseXY`(10,`GetMouseY`);

If (`GetMouseY`>20) **then**

`SetMouseXY`(`GetMouseX`, 5);

Until (`GetMouseButtons`=`MouseRightButton`);

`DoneMouse`;

end.

ShowMouse

Synopsis: Show the mouse cursor.

Declaration: `procedure ShowMouse`

Visibility: default

Description: `ShowMouse` shows the mouse cursor if it was previously hidden. The capability to hide or show the mouse cursor depends on the driver.

For an example, see `HideMouse` ([651](#))

Errors: None.

See also: `HideMouse` ([651](#))

Chapter 20

Reference for unit 'Objects'

20.1 Overview

This document documents the `objects` unit. The unit was implemented by many people, and was mainly taken from the FreeVision sources. It has been ported to all supported platforms.

The methods and fields that are in a `Private` part of an object declaration have been left out of this documentation.

20.2 Constants, types and variables

Constants

`coIndexError = -1`

Collection list error: Index out of range

`coOverflow = -2`

Collection list error: Overflow

`DefaultTPCompatible : Boolean = false`

`MaxBytes = 128 * 1024 * 128`

Maximum data size (in bytes)

`MaxCollectionSize = MaxBytes div SizeOf (Pointer)`

Maximum collection size (in items)

`MaxPtrs = MaxBytes div SizeOf (Pointer)`

Maximum data size (in pointers)

`MaxReadBytes = $7fffffff`

Maximum data that can be read from a stream (not used)

MaxTPCompatibleCollectionSize = 65520 div 4

Maximum collection size (in items, same value as in TP)

MaxWords = MaxBytes div SizeOf (Word)

Maximum data size (in words)

RCollection : TStreamRec = (ObjType:50;VmtLink:Ofs (TypeOf (TCollection) ^) ;Load

Default stream record for the TCollection (670) object.

RStrCollection : TStreamRec = (ObjType:69;VmtLink:Ofs (TypeOf (TStrCollection) ^) ;Load

Default stream record for the TStrCollection (710) object.

RStringCollection : TStreamRec = (ObjType:51;VmtLink:Ofs (TypeOf (TStringCollection) ^) ;Load

Default stream record for the TStringCollection (720) object.

RStringList : TStreamRec = (ObjType:52;VmtLink:Ofs (TypeOf (TStringList) ^) ;Load

Default stream record for the TStringList (722) object.

RStrListMaker : TStreamRec = (ObjType:52;VmtLink:Ofs (TypeOf (TStrListMaker) ^) ;Load

Default stream record for the TStrListMaker (724) object.

stCreate = \$3C00

Stream initialization mode: Create new file

stError = -1

Stream error codes: Access error

stGetError = -5

Stream error codes: Get object error

stInitError = -2

Stream error codes: Initialize error

stOk = 0

Stream error codes: No error

stOpen = \$3D02

Stream initialization mode: Read/write access

stOpenError = -8

Stream error codes: Error opening stream

`stOpenRead = $3D00`

Stream initialization mode: Read access only

`stOpenWrite = $3D01`

Stream initialization mode: Write access only

`stPutError = -6`

Stream error codes: Put object error

`stReadError = -3`

Stream error codes: Stream read error

`StreamError : Pointer = nil`

Pointer to default stream error handler.

`stSeekError = -7`

Stream error codes: Seek error in stream

`stWriteError = -4`

Stream error codes: Stream write error

`vmtHeaderSize = 8`

Size of the VMT header in an object (not used).

Types

`AsciiZ = Array[0..255] of Char`

Filename - null terminated array of characters.

`FNameStr = String`

Filename - shortstring version.

`LongRec = packed record`

`Hi : Word;`

`Lo : Word;`

`end`

Record describing a longint (in Words)

`PBufStream = ^TBufStream`

Pointer to TBufStream (666) object.

PByteArray = ^TByteArray

Pointer to TByteArray (659)

PCharSet = ^TCharSet

Pointer to TCharSet (660).

PCollection = ^TCollection

Pointer to TCollection (670) object.

PDosStream = ^TDosStream

Pointer to TDosStream (685) object.

PItemList = ^TItemList

Pointer to TItemList (660) object.

PMemoryStream = ^TMemoryStream

Pointer to TMemoryStream (690) object.

PObject = ^TObject

Pointer to TObject (692) object.

PPoint = ^TPoint

Pointer to TPoint (694) record.

PPointerArray = ^TPointerArray

Pointer to TPointerArray (660)

PRect = ^TRect

Pointer to TRect (694) object.

PResourceCollection = ^TResourceCollection

Pointer to TResourceCollection (700) object.

PResourceFile = ^TResourceFile

Pointer to TResourceFile (701) object.

PSortedCollection = ^TSortedCollection

Pointer to TSortedCollection (704) object.

PStrCollection = ^TStrCollection

Pointer to TStrCollection (710) object.

`PStream = ^TStream`

Pointer type to `TStream` (712)

`PStreamRec = ^TStreamRec`

Pointer to `TStreamRec` (660)

`PStrIndex = ^TStrIndex`

Pointer to `TStrIndex` (660) array.

`PString = PShortString`

Pointer to a shortstring.

`PStringCollection = ^TStringCollection`

Pointer to `TStringCollection` (720) object.

`PStringList = ^TStringList`

Pointer to `TStringList` (722) object.

`PStrListMaker = ^TStrListMaker`

Pointer to `TStrListMaker` (724) object.

```
PtrRec = packed record
  Ofs : Word;
  Seg : Word;
end
```

Record describing a pointer to a memory location.

`PUnSortedStrCollection = ^TUnSortedStrCollection`

Pointer to `TUnsortedStrCollection` (725) object.

`PWordArray = ^TWordArray`

Pointer to `TWordArray` (660)

`Sw_Integer = LongInt`

Alias for `longint`

`Sw_Word = Cardinal`

Alias for `Cardinal`

`TByteArray = Array[0..MaxBytes-1] of Byte`

Array with maximum allowed number of bytes.

TCharSet = Set of Char

Generic set of characters type.

TItemList = Array[0..MaxCollectionSize-1] of Pointer

Pointer array type used in a TCollection ([670](#))

TPointerArray = Array[0..MaxPtrs-1] of Pointer

Array with maximum allowed number of pointers

```
TStreamRec = packed record
  ObjType : Sw_Word;
  VmtLink : pointer;
  Load : Pointer;
  Store : Pointer;
  Next : PStreamRec;
end
```

TStreamRec is used by the **Objects** unit streaming mechanism: when an object is registered, a TStreamRec record is added to a list of records. This list is used when objects need to be streamed from/streamed to a stream. It contains all the information needed to stream the object.

TStrIndex = Array[0..9999] of TStrIndexRec

Pointer array type used in a TStringList ([722](#))

```
TStrIndexRec = packed record
  Key : Sw_Word;
  Count : Word;
  Offset : Word;
end
```

Record type used in a TStringList ([722](#)) to store the strings

TWordArray = Array[0..MaxWords-1] of Word

Array with maximum allowed number of words.

```
WordRec = packed record
  Hi : Byte;
  Lo : Byte;
end
```

Record describing a Word (in bytes)

Variables

invalidhandle : THandle

Value for invalid handle. Initial value for file stream handles or when the stream is closed.

20.3 Procedures and functions

Abstract

Synopsis: Abstract error handler.

Declaration: `procedure Abstract`

Visibility: default

Description: When implementing abstract methods, do not declare them as `abstract`. Instead, define them simply as `virtual`. In the implementation of such abstract methods, call the `Abstract` procedure. This allows explicit control of what happens when an abstract method is called.

The current implementation of `Abstract` terminates the program with a run-time error 211.

Errors: None.

CallPointerConstructor

Declaration: `function CallPointerConstructor(Ctor: pointer;Obj: pointer;VMT: pointer;
Param1: pointer) : pointer`

Visibility: default

CallPointerLocal

Declaration: `function CallPointerLocal(Func: pointer;Frame: Pointer;Param1: pointer)
: pointer`

Visibility: default

CallPointerMethod

Declaration: `function CallPointerMethod(Method: pointer;Obj: pointer;Param1: pointer)
: pointer`

Visibility: default

CallPointerMethodLocal

Declaration: `function CallPointerMethodLocal(Func: pointer;Frame: Pointer;
Obj: pointer;Param1: pointer) : pointer`

Visibility: default

CallVoidConstructor

Declaration: `function CallVoidConstructor(Ctor: pointer;Obj: pointer;VMT: pointer)
: pointer`

Visibility: default

CallVoidLocal

Declaration: `function CallVoidLocal(Func: pointer;Frame: Pointer) : pointer`

Visibility: default

CallVoidMethod

Declaration: `function CallVoidMethod(Method: pointer;Obj: pointer) : pointer`

Visibility: default

CallVoidMethodLocal

Declaration: `function CallVoidMethodLocal(Func: pointer;Frame: Pointer;Obj: pointer)
: pointer`

Visibility: default

DisposeStr

Synopsis: Dispose of a shortstring which was allocated on the heap.

Declaration: `procedure DisposeStr(P: PString)`

Visibility: default

Description: `DisposeStr` removes a dynamically allocated string from the heap.

For an example, see `NewStr` ([663](#)).

Errors: None.

See also: `NewStr` ([663](#)), `SetStr` ([665](#))

LongDiv

Synopsis: Overflow safe divide

Declaration: `function LongDiv(X: LongInt;Y: Integer) : Integer`

Visibility: default

Description: `LongDiv` divides X by Y. The result is of type `Integer` instead of type `Longint`, as you would get normally.

Errors: If Y is zero, a run-time error will be generated.

See also: `LongMul` ([662](#))

LongMul

Synopsis: Overflow safe multiply.

Declaration: `function LongMul(X: Integer;Y: Integer) : LongInt`

Visibility: default

Description: LongMul multiplies X with Y. The result is of type Longint. This avoids possible overflow errors you would normally get when multiplying X and Y that are too big.

Errors: None.

See also: LongDiv ([662](#))

NewStr

Synopsis: Allocate a copy of a shortstring on the heap.

Declaration: function NewStr(const S: String) : PString

Visibility: default

Description: NewStr makes a copy of the string S on the heap, and returns a pointer to this copy. If the string is empty then Nil is returned.

The allocated memory is not based on the declared size of the string passed to NewStr, but is based on the actual length of the string.

Errors: If not enough memory is available, an 'out of memory' error will occur.

See also: DisposeStr ([662](#)), SetStr ([665](#))

Listing: ./objectex/ex40.pp

```

Program ex40;

{ Program to demonstrate the NewStr function }

Uses Objects;

Var S : String;
    P : PString;

begin
  S:= 'Some really cute string';
  WriteLn ( 'Memavail : ',Memavail);
  P:=NewStr(S);
  If P^<>S then
    WriteLn ( 'Oh-oh... Something is wrong !! ');
    WriteLn ( 'Allocated string. Memavail : ',Memavail);
    DisposeStr(P);
    WriteLn ( 'Deallocated string. Memavail : ',Memavail);
end.

```

RegisterObjects

Synopsis: Register standard objects.

Declaration: procedure RegisterObjects

Visibility: default

Description: RegisterObjects registers the following objects for streaming:

1.TCollection, see TCollection ([670](#)).

2.TStringCollection, see TStringCollection (720).

3.TStrCollection, see TStrCollection (710).

Errors: None.

See also: RegisterType (664)

RegisterType

Synopsis: Register new object for streaming.

Declaration: `procedure RegisterType(var S: TStreamRec)`

Visibility: default

Description: RegisterType registers a new type for streaming. An object cannot be streamed unless it has been registered first. The stream record S needs to have the following fields set:

ObjType: Sw_WordThis should be a unique identifier. Each possible type should have it's own identifier.

VmtLink: pointerThis should contain a pointer to the VMT (Virtual Method Table) of the object you try to register.

Load : Pointeris a pointer to a method that initializes an instance of that object, and reads the initial values from a stream. This method should accept as it's sole argument a PStream type variable.

Store: Pointeris a pointer to a method that stores an instance of the object to a stream. This method should accept as it's sole argument a PStream type variable.

The VMT of the object can be retrieved with the following expression:

```
VmtLink: ofs(EOF(MyType)^);
```

Errors: In case of error (if a object with the same ObjType) is already registered), run-time error 212 occurs.

Listing: ./objectex/myobject.pp

Unit MyObject;

Interface

Uses Objects;

Type

```
PMyObject = ^TMyObject;
TMyObject = Object(TObject)
  Field : Longint;
  Constructor Init;
  Constructor Load (Var Stream : TStream);
  Destructor Done;
  Procedure Store (Var Stream : TStream);
  Function GetField : Longint;
  Procedure SetField (Value : Longint);
end;
```

Implementation

```

Constructor TMyobject.Init;

begin
    Inherited Init;
    Field := -1;
end;

Constructor TMyobject.Load (Var Stream : TStream);

begin
    Stream.Read( Field , Sizeof( Field ));
end;

Destructor TMyObject.Done;

begin
end;

Function TMyObject.GetField : Longint;

begin
    GetField := Field;
end;

Procedure TMyObject.SetField (Value : Longint);

begin
    Field := Value;
end;

Procedure TMyObject.Store (Var Stream : TStream);

begin
    Stream.Write( Field , SizeOf( Field ));
end;

Const MyObjectRec : TStreamRec = (
    Objtype : 666;
    vmtlink : Ofs(TypeOf(TMyObject)^);
    Load : @TMyObject.Load;
    Store : @TMyObject.Store;
);

begin
    RegisterObjects;
    RegisterType ( MyObjectRec );
end.

```

SetStr

Synopsis: Allocate a copy of a shortstring on the heap.

Declaration: procedure SetStr(var p: PString; const s: String)

Visibility: default

Description: `SetStr` makes a copy of the string `S` on the heap and returns the pointer to this copy in `P`. If `P` pointed to another string (i.e. was not `Nil`, the memory is released first. Contrary to `NewStr` (663), if the string is empty then a pointer to an empty string is returned.

The allocated memory is not based on the declared size of the string passed to `NewStr`, but is based on the actual length of the string.

Errors: If not enough memory is available, an 'out of memory' error will occur.

See also: `DisposeStr` (662), `NewStr` (663)

20.4 TBufStream

Description

`Bufstream` implements a buffered file stream. That is, all data written to the stream is written to memory first. Only when the buffer is full, or on explicit request, the data is written to disk.

Also, when reading from the stream, first the buffer is checked if there is any unread data in it. If so, this is read first. If not the buffer is filled again, and then the data is read from the buffer.

The size of the buffer is fixed and is set when constructing the file.

This is useful if you need heavy throughput for your stream, because it speeds up operations.

Method overview

Page	Method	Description
667	<code>Close</code>	Flush data and Close the file.
667	<code>Done</code>	Close the file and cleans up the instance.
667	<code>Flush</code>	FLush data from buffer, and write it to stream.
666	<code>Init</code>	Initialize an instance of <code>TBufStream</code> and open the file.
669	<code>Open</code>	Open the file if it is closed.
669	<code>Read</code>	Read data from the file to a buffer in memory.
668	<code>Seek</code>	Set current position in file.
668	<code>Truncate</code>	Flush buffer, and truncate the file at current position.
669	<code>Write</code>	Write data to the file from a buffer in memory.

TBufStream.Init

Synopsis: Initialize an instance of `TBufStream` and open the file.

Declaration: `constructor Init(FileName: FNameStr;Mode: Word;Size: Word)`

Visibility: default

Description: `Init` instantiates an instance of `TBufStream`. The name of the file that contains (or will contain) the data of the stream is given in `FileName`. The `Mode` parameter determines whether a new file should be created and what access rights you have on the file. It can be one of the following constants:

stCreateCreates a new file.

stOpenReadRead access only.

stOpenWriteWrite access only.

stOpenRead and write access.

The `Size` parameter determines the size of the buffer that will be created. It should be different from zero.

For an example see `TBufStream.Flush` (667).

Errors: On error, `Status` is set to `stInitError`, and `ErrorInfo` is set to the dos error code.

See also: `TDosStream.Init` (686), `TBufStream.Done` (667)

TBufStream.Done

Synopsis: Close the file and cleans up the instance.

Declaration: `destructor Done; Virtual`

Visibility: `default`

Description: `Done` flushes and closes the file if it was open and cleans up the instance of `TBufStream`.

For an example see `TBufStream.Flush` (667).

Errors: None.

See also: `TDosStream.Done` (686), `TBufStream.Init` (666), `TBufStream.Close` (667)

TBufStream.Close

Synopsis: Flush data and Close the file.

Declaration: `procedure Close; Virtual`

Visibility: `default`

Description: `Close` flushes and closes the file if it was open, and sets `Handle` to -1. Contrary to `Done` (667) it does not clean up the instance of `TBufStream`

For an example see `TBufStream.Flush` (667).

Errors: None.

See also: `TStream.Close` (716), `TBufStream.Init` (666), `TBufStream.Done` (667)

TBufStream.Flush

Synopsis: FLush data from buffer, and write it to stream.

Declaration: `procedure Flush; Virtual`

Visibility: `default`

Description: When the stream is in write mode, the contents of the buffer are written to disk, and the buffer position is set to zero. When the stream is in read mode, the buffer position is set to zero.

Errors: Write errors may occur if the file was in write mode. see `Write` (669) for more info on the errors.

See also: `TStream.Close` (716), `TBufStream.Init` (666), `TBufStream.Done` (667)

Listing: `./objectex/ex15.pp`

```

Program ex15;

{ Program to demonstrate the TStream.Flush method }

Uses Objects;

Var L : String;
    P : PString;
    S : PBufStream; { Only one with Flush implemented. }

begin
    L:= 'Some constant string';
    { Buffer size of 100 }
    S:=New(PBufStream, Init('test.dat', stcreate, 100));
    WriteLn ('Writing "', L, '" to stream with handle ', S^.Handle);
    S^.WriteStr(@L);
    { At this moment, there is no data on disk yet. }
    S^.Flush;
    { Now there is. }
    S^.WriteStr(@L);
    { Close calls flush first }
    S^.Close;
    WriteLn ('Closed stream. File handle is ', S^.Handle);
    S^.Open(stOpenRead);
    P:=S^.ReadStr;
    L:=P^;
    DisposeStr(P);
    WriteLn ('Read "', L, '" from stream with handle ', S^.Handle);
    S^.Close;
    Dispose (S, Done);
end.

```

TBufStream.Truncate

Synopsis: Flush buffer, and truncate the file at current position.

Declaration: `procedure Truncate; Virtual`

Visibility: default

Description: If the status of the stream is `stOK`, then `Truncate` tries to flush the buffer, and then truncates the stream size to the current file position.

For an example, see `TDosStream.Truncate` (687).

Errors: Errors can be those of `Flush` (667) or `TDosStream.Truncate` (687).

See also: `TStream.Truncate` (717), `TDosStream.Truncate` (687), `TStream.GetSize` (714)

TBufStream.Seek

Synopsis: Set current position in file.

Declaration: `procedure Seek(Pos: LongInt); Virtual`

Visibility: default

Description: If the stream's status is `stOK`, then `Seek` sets the file position to `Pos`. `Pos` is a zero-based offset, counted from the beginning of the file.

For an example, see `TStream.Seek` (718);

Errors: In case an error occurs, the stream's status is set to `stSeekError`, and the OS error code is stored in `ErrorInfo`.

See also: `TStream.Seek` (718), `TStream.GetPos` (714)

TBufStream.Open

Synopsis: Open the file if it is closed.

Declaration: `procedure Open(OpenMode: Word); Virtual`

Visibility: `default`

Description: If the stream's status is `stOK`, and the stream is closed then `Open` re-opens the file stream with mode `OpenMode`. This call can be used after a `Close` (667) call.

For an example, see `TDosStream.Open` (688).

Errors: If an error occurs when re-opening the file, then `Status` is set to `stOpenError`, and the OS error code is stored in `ErrorInfo`

See also: `TStream.Open` (716), `TBufStream.Close` (667)

TBufStream.Read

Synopsis: Read data from the file to a buffer in memory.

Declaration: `procedure Read(var Buf; Count: LongInt); Virtual`

Visibility: `default`

Description: If the Stream is open and the stream status is `stOK` then `Read` will read `Count` bytes from the stream and place them in `Buf`.

`Read` will first try to read the data from the stream's internal buffer. If insufficient data is available, the buffer will be filled before continuing to read. This process is repeated until all needed data has been read.

For an example, see `TStream.Read` (719).

Errors: In case of an error, `Status` is set to `StReadError`, and `ErrorInfo` gets the OS specific error, or 0 when an attempt was made to read beyond the end of the stream.

See also: `TStream.Read` (719), `TBufStream.Write` (669)

TBufStream.Write

Synopsis: Write data to the file from a buffer in memory.

Declaration: `procedure Write(var Buf; Count: LongInt); Virtual`

Visibility: `default`

Description: If the Stream is open and the stream status is `stOK` then `Write` will write `Count` bytes from `Buf` and place them in the stream.

`Write` will first try to write the data to the stream's internal buffer. When the internal buffer is full, then the contents will be written to disk. This process is repeated until all data has been written.

For an example, see `TStream.Read` (719).

Errors: In case of an error, `Status` is set to `StWriteError`, and `ErrorInfo` gets the OS specific error.

See also: `TStream.Write` (719), `TBufStream.Read` (669)

20.5 TCollection

Description

The `TCollection` object manages a collection of pointers or objects. It also provides a series of methods to manipulate these pointers or objects.

Whether or not objects are used depends on the kind of calls you use. All kinds come in 2 flavors, one for objects, one for pointers.

Method overview

Page	Method	Description
672	<code>At</code>	Return the item at a certain index.
681	<code>AtDelete</code>	Delete item at certain position.
680	<code>AtFree</code>	Free an item at the indicates position, calling it's destructor.
684	<code>AtInsert</code>	Insert an element at a certain position in the collection.
683	<code>AtPut</code>	Set collection item, overwriting an existing value.
679	<code>Delete</code>	Delete an item from the collection, but does not destroy it.
677	<code>DeleteAll</code>	Delete all elements from the collection. Objects are not destroyed.
672	<code>Done</code>	Clean up collection, release all memory.
683	<code>Error</code>	Set error code.
675	<code>FirstThat</code>	Return first item which matches a test.
682	<code>ForEach</code>	Execute procedure for each item in the list.
678	<code>Free</code>	Free item from collection, calling it's destructor.
676	<code>FreeAll</code>	Release all objects from the collection.
681	<code>FreeItem</code>	Destroy a non-nil item.
673	<code>GetItem</code>	Read one item off the stream.
673	<code>IndexOf</code>	Find the position of a certain item.
670	<code>Init</code>	Instantiate a new collection.
679	<code>Insert</code>	Insert a new item in the collection at the end.
674	<code>LastThat</code>	Return last item which matches a test.
671	<code>Load</code>	Initialize a new collection and load collection from a stream.
675	<code>Pack</code>	Remove all <code>>Nil</code> pointers from the collection.
685	<code>PutItem</code>	Put one item on the stream
683	<code>SetLimit</code>	Set maximum number of elements in the collection.
684	<code>Store</code>	Write collection to a stream.

TCollection.Init

Synopsis: Instantiate a new collection.

Declaration: constructor `Init(ALimit: Sw_Integer; ADelta: Sw_Integer)`

Visibility: default

Description: `Init` initializes a new instance of a collection. It sets the (initial) maximum number of items in the collection to `ALimit`. `ADelta` is the increase size : The number of memory places that will be allocated in case `ALimit` is reached, and another element is added to the collection.

For an example, see `TCollection.ForEach` (682).

Errors: None.

See also: `TCollection.Load` (671), `TCollection.Done` (672)

TCollection.Load

Synopsis: Initialize a new collection and load collection from a stream.

Declaration: constructor `Load(var S: TStream)`

Visibility: default

Description: `Load` initializes a new instance of a collection. It reads from stream `S` the item count, the item limit count, and the increase size. After that, it reads the specified number of items from the stream.

Errors: Errors returned can be those of `GetItem` (673).

See also: `TCollection.Init` (670), `TCollection.GetItem` (673), `TCollection.Done` (672)

Listing: `./objectex/ex22.pp`

Program `ex22`;

{ Program to demonstrate the TCollection.Load method }

Uses `Objects, MyObject`; *{ For TMyObject definition and registration }*

Var `C : PCollection`;
 `M : PMyObject`;
 `I : Longint`;
 `S : PMemoryStream`;

begin

`C:=New(PCollection, Init(100,10));`

For `I:=1 to 100 do`

begin

`M:=New(PMyObject, Init);`

`M^.SetField(100-I);`

`C^.Insert(M);`

end;

WriteLn ('Inserted ', `C^.Count`, ' objects ');

`S:=New(PMemoryStream, Init(1000,10));`

`C^.Store(S^);`

`C^.FreeAll;`

Dispose(`C`, `Done`);

`S^.Seek(0);`

`C^.Load(S^);`

WriteLn ('Read ', `C^.Count`, ' objects from stream.');

Dispose(`S`, `Done`);

Dispose(`C`, `Done`);

end.

TCollection.Done

Synopsis: Clean up collection, release all memory.

Declaration: `destructor Done; Virtual`

Visibility: `default`

Description: Done frees all objects in the collection, and then releases all memory occupied by the instance.

For an example, see TCollection.ForEach ([682](#)).

Errors: None.

See also: TCollection.Init ([670](#)), TCollection.FreeAll ([676](#))

TCollection.At

Synopsis: Return the item at a certain index.

Declaration: `function At(Index: Sw_Integer) : Pointer`

Visibility: `default`

Description: At returns the item at position Index.

Errors: If Index is less than zero or larger than the number of items in the collection, `seep{Error}{TCollection.Error}` is called with `coIndexError` and Index as arguments, resulting in a run-time error.

See also: TCollection.Insert ([679](#))

Listing: `./objectex/ex23.pp`

Program `ex23;`

{ Program to demonstrate the TCollection.At method }

Uses `Objects, MyObject; { For TMyObject definition and registration }`

Var `C : PCollection;
M : PMyObject;
I : Longint;`

begin

`C:=New(PCollection, Init(100,10));`

For `I:=1 to 100 do`

begin

`M:=New(PMyObject, Init);`

`M^.SetField(100-I);`

`C^.Insert(M);`

end;

For `I:=0 to C^.Count-1 do`

begin

`M:=C^.At(I);`

`Writeln('Object ', i, ' has field : ', M^.GetField);`

end;

`C^.FreeAll;`

Dispose `(C, Done);`

end.

TCollection.IndexOf

Synopsis: Find the position of a certain item.

Declaration: `function IndexOf(Item: Pointer) : Sw_Integer; Virtual`

Visibility: default

Description: `IndexOf` returns the index of `Item` in the collection. If `Item` isn't present in the collection, -1 is returned.

Errors: If the item is not present, -1 is returned.

See also: `TCollection.At` (672), `TCollection.GetItem` (673), `TCollection.Insert` (679)

Listing: `./objectex/ex24.pp`

Program `ex24;`

{ Program to demonstrate the TCollection.IndexOf method }

Uses `Objects, MyObject; { For TMyObject definition and registration }`

Var `C : PCollection;`
 `M, Keep : PMyObject;`
 `I : Longint;`

begin

`Randomize;`

`C:=New(PCollection, Init(100,10));`

`Keep:=Nil;`

For `I:=1 to 100 do`

begin

`M:=New(PMyObject, Init);`

`M^.SetField(I-1);`

If `Random<0.1 then`

`Keep:=M;`

`C^.Insert(M);`

end;

If `Keep=Nil then`

begin

`Writeln('Please run again. No object selected');`

`Halt(1);`

end;

`Writeln('Selected object has field : ', Keep^.GetField);`

`Write('Selected object has index : ', C^.IndexOf(Keep));`

`Writeln(' should match it's field.');`

`C^.FreeAll;`

`Dispose(C, Done);`

end.

TCollection.GetItem

Synopsis: Read one item off the stream.

Declaration: `function GetItem(var S: TStream) : Pointer; Virtual`

Visibility: default

Description: `GetItem` reads a single item off the stream `S`, and returns a pointer to this item. This method is used internally by the `Load` method, and should not be used directly.

Errors: Possible errors are the ones from `TStream.Get` ([712](#)).

See also: `TStream.Get` ([712](#)), `TCollection.Store` ([684](#))

TCollection.LastThat

Synopsis: Return last item which matches a test.

Declaration: `function LastThat(Test: Pointer) : Pointer`

Visibility: default

Description: This function returns the last item in the collection for which `Test` returns a non-nil result. `Test` is a function that accepts 1 argument: a pointer to an object, and that returns a pointer as a result.

Errors: None.

See also: `TCollection.FirstThat` ([675](#))

Listing: `./objectex/ex25.pp`

Program `ex21`;

{ Program to demonstrate the TCollection.Foreach method }

Uses `Objects, MyObject`; *{ For TMyObject definition and registration }*

Var `C : PCollection`;
 `M : PMyObject`;
 `I : Longint`;

Function `CheckField (Dummy: Pointer; P : PMyObject) : Longint`;

begin
 If `P^.GetField < 56` **then**
 `Checkfield := 1`
 else
 `CheckField := 0`;
end;

begin
 `C := New (PCollection, Init (100, 10));`
 For `I := 1 to 100` **do**
 begin
 `M := New (PMyObject, Init);`
 `M^.SetField (I);`
 `C^.Insert (M);`
 end;
 Writeln ('Inserted ', `C^.Count`, ' objects ');
 Writeln ('Last one for which Field < 56 has index (should be 54) : ',
 `C^.IndexOf (C^.LastThat (@CheckField))`);
 `C^.FreeAll`;
 Dispose (`C`, `Done`);
end.

TCollection.FirstThat

Synopsis: Return first item which matches a test.

Declaration: `function FirstThat(Test: Pointer) : Pointer`

Visibility: default

Description: This function returns the first item in the collection for which `Test` returns a non-nil result. `Test` is a function that accepts 1 argument: a pointer to an object, and that returns a pointer as a result.

Errors: None.

See also: `TCollection.LastThat` ([674](#))

Listing: `./objectex/ex26.pp`

Program `ex21`;

{ Program to demonstrate the TCollection.FirstThat method }

Uses `Objects, MyObject`; *{ For TMyObject definition and registration }*

Var `C : PCollection`;
 `M : PMyObject`;
 `I : Longint`;

Function `CheckField (Dummy: Pointer; P : PMyObject) : Longint`;

begin
 If `P^.GetField > 56` **then**
 `Checkfield := 1`
 else
 `CheckField := 0`;
end;

begin
 `C := New (PCollection, Init (100, 10));`
 For `I := 1` **to** `100` **do**
 begin
 `M := New (PMyObject, Init);`
 `M^.SetField (I);`
 `C^.Insert (M);`
 end;
 WriteLn ('Inserted ', `C^.Count`, ' objects ');
 WriteLn ('first one for which Field > 56 has index (should be 56) : ',
 `C^.IndexOf (C^.FirstThat (@CheckField))`);
 `C^.FreeAll`;
 Dispose (`C`, `Done`);
end.

TCollection.Pack

Synopsis: Remove all `>Nil` pointers from the collection.

Declaration: `procedure Pack`

Visibility: default

Description: Pack removes all Nil pointers from the collection, and adjusts Count to reflect this change. No memory is freed as a result of this call. In order to free any memory, you can call SetLimit with an argument of Count after a call to Pack.

Errors: None.

See also: TCollection.SetLimit ([683](#))

Listing: ./objectex/ex26.pp

Program ex21;

{ Program to demonstrate the TCollection.FirstThat method }

Uses Objects, MyObject; *{ For TMyObject definition and registration }*

Var C : PCollection;
 M : PMyObject;
 I : Longint;

Function CheckField (Dummy: Pointer; P : PMyObject) : Longint;

begin
 If P^.GetField > 56 **then**
 Checkfield := 1
 else
 CheckField := 0;
end;

begin
 C := **New**(PCollection, Init(100, 10));
 For I := 1 **to** 100 **do**
 begin
 M := **New**(PMyObject, Init);
 M^.SetField(I);
 C^.Insert(M);
 end;
 Writeln ('Inserted ', C^.Count, ' objects');
 Writeln ('first one for which Field > 56 has index (should be 56) : ',
 C^.IndexOf(C^.FirstThat(@CheckField)));
 C^.FreeAll;
 Dispose(C, Done);
end.

TCollection.FreeAll

Synopsis: Release all objects from the collection.

Declaration: procedure FreeAll

Visibility: default

Description: FreeAll calls the destructor of each object in the collection. It doesn't release any memory occupied by the collection itself, but it does set Count to zero.

Errors:

See also: TCollection.DeleteAll ([677](#)), TCollection.FreeItem ([681](#))

Listing: ./objectex/ex28.pp

```

Program ex28;

{ Program to demonstrate the TCollection.FreeAll method }

Uses Objects, MyObject; { For TMyObject definition and registration }

Var C : PCollection;
    M : PMyObject;
    I, InitMem : Longint;

begin
  Randomize;
  C:=New(PCollection, Init(120,10));
  InitMem:=Memavail;
  Writeln ( 'Initial memory : ', InitMem);
  For I:=1 to 100 do
    begin
      M:=New(PMyObject, Init);
      M^.SetField(I-1);
      C^.Insert(M);
    end;
    Writeln ( 'Added 100 Items. Memory available : ', Memavail);
    Write ( 'Lost : ', Initmem-Memavail, ' bytes. ');
    Write ( ' (Should be 100* ', SizeOf(TMyObject));
    Writeln ( '=', 100*SizeOf(TMyObject), ' ) ');
    C^.FreeAll;
    Writeln ( 'Freed all objects. Memory available : ', Memavail);
    Writeln ( 'Lost : ', Initmem-Memavail, ' bytes. ');
    Dispose(C, Done);
end.

```

TCollection.DeleteAll

Synopsis: Delete all elements from the collection. Objects are not destroyed.

Declaration: procedure DeleteAll

Visibility: default

Description: DeleteAll deletes all elements from the collection. It just sets the Count variable to zero. Contrary to FreeAll (676), DeleteAll doesn't call the destructor of the objects.

Errors: None.

See also: TCollection.FreeAll (676), TCollection.Delete (679)

Listing: ./objectex/ex29.pp

```

Program ex29;

{
  Program to demonstrate the TCollection.DeleteAll method
  Compare with example 28, where FreeAll is used.
}

Uses Objects, MyObject; { For TMyObject definition and registration }

```

```

Var C : PCollection;
      M : PMyObject;
      I, InitMem : Longint;

begin
  Randomize;
  C:=New(PCollection, Init(120,10));
  InitMem:=Memavail;
  Writeln ('Initial memory : ', InitMem);
  For I:=1 to 100 do
    begin
      M:=New(PMyObject, Init);
      M^.SetField(I-1);
      C^.Insert(M);
    end;
    Writeln ('Added 100 Items. Memory available : ', Memavail);
    Write ('Lost : ', Initmem-Memavail, ' bytes. ');
    Write (' (Should be 100* ', SizeOf(TMyObject));
    Writeln ('=', 100*SizeOf(TMyObject), ') ');
    C^.DeleteAll;
    Writeln ('Deleted all objects. Memory available : ', Memavail);
    Writeln ('Lost : ', Initmem-Memavail, ' bytes. ');
    Dispose(C, Done);
end.

```

TCollection.Free

Synopsis: Free item from collection, calling it's destructor.

Declaration: `procedure Free(Item: Pointer)`

Visibility: default

Description: Free Deletes Item from the collection, and calls the destructor Done of the object.

Errors: If the Item is not in the collection, Error will be called with `coIndexError`.

See also: `TCollection.FreeItem` ([681](#))

Listing: `./objectex/ex30.pp`

```

Program ex30;

{ Program to demonstrate the TCollection.Free method }

Uses Objects, MyObject; { For TMyObject definition and registration }

Var C : PCollection;
      M : PMyObject;
      I, InitMem : Longint;

begin
  Randomize;
  C:=New(PCollection, Init(120,10));
  InitMem:=Memavail;
  Writeln ('Initial memory : ', InitMem);
  For I:=1 to 100 do

```

```

    begin
    M:=New(PMyObject, Init);
    M^.SetField(I-1);
    C^.Insert(M);
    end;
    Writeln ('Added 100 Items. Memory available : ',Memavail);
    Write ('Lost : ',Initmem-Memavail, ' bytes. ');
    Write (' (Should be 100*',SizeOf(TMyObject));
    Writeln ('=',100*SizeOf(TMyObject),') ');
    With C^ do
        While Count>0 do Free(At(Count-1));
    Writeln ('Freed all objects. Memory available : ',Memavail);
    Writeln ('Lost : ',Initmem-Memavail, ' bytes. ');
    Dispose(C,Done);
end.

```

TCollection.Insert

Synopsis: Insert a new item in the collection at the end.

Declaration: `procedure Insert(Item: Pointer); Virtual`

Visibility: default

Description: `Insert` inserts `Item` in the collection. `TCollection` inserts this item at the end, but descendent objects may insert it at another place.

Errors: None.

See also: `TCollection.AtInsert` (684), `TCollection.AtPut` (683)

TCollection.Delete

Synopsis: Delete an item from the collection, but does not destroy it.

Declaration: `procedure Delete(Item: Pointer)`

Visibility: default

Description: `Delete` deletes `Item` from the collection. It doesn't call the item's destructor, though. For this the `Free` (678) call is provided.

Errors: If the `Item` is not in the collection, `Error` will be called with `coIndexError`.

See also: `TCollection.AtDelete` (681), `TCollection.Free` (678)

Listing: `./objectex/ex31.pp`

Program `ex31`;

{ Program to demonstrate the TCollection.Delete method }

Uses `Objects, MyObject`; *{ For TMyObject definition and registration }*

Var `C : PCollection`;
 `M : PMyObject`;
 `I, InitMem : Longint`;

```

begin
  Randomize;
  C:=New(PCollection, Init(120,10));
  InitMem:=Memavail;
  Writeln('Initial memory : ', InitMem);
  For I:=1 to 100 do
    begin
      M:=New(PMyObject, Init);
      M^.SetField(I-1);
      C^.Insert(M);
    end;
  Writeln('Added 100 Items. Memory available : ', Memavail);
  Write('Lost : ', Initmem-Memavail, ' bytes. ');
  Write(' (Should be 100*', SizeOf(TMyObject));
  Writeln('=', 100*SizeOf(TMyObject), ') ');
  With C^ do
    While Count>0 do Delete(At(Count-1));
  Writeln('Freed all objects. Memory available : ', Memavail);
  Writeln('Lost : ', Initmem-Memavail, ' bytes. ');
  Dispose(C, Done);
end.

```

TCollection.AtFree

Synopsis: Free an item at the indicates position, calling it's destructor.

Declaration: procedure AtFree(Index: Sw_Integer)

Visibility: default

Description: AtFree deletes the item at position Index in the collection, and calls the item's destructor if it is not Nil.

Errors: If Index isn't valid then Error (683) is called with CoIndexError.

See also: TCollection.Free (678), TCollection.AtDelete (681)

Listing: ./objectex/ex32.pp

Program ex32;

{ Program to demonstrate the TCollection.AtFree method }

Uses Objects, MyObject; *{ For TMyObject definition and registration }*

Var C : PCollection;
 M : PMyObject;
 I, InitMem : Longint;

```

begin
  Randomize;
  C:=New(PCollection, Init(120,10));
  InitMem:=Memavail;
  Writeln('Initial memory : ', InitMem);
  For I:=1 to 100 do
    begin
      M:=New(PMyObject, Init);
      M^.SetField(I-1);

```

```

    C^.Insert(M);
  end;
  Writeln ('Added 100 Items. Memory available : ',Memavail);
  Write ('Lost : ',Initmem-Memavail,' bytes. ');
  Write ('(Should be 100*',SizeOf(TMyObject));
  Writeln ('=',100*SizeOf(TMyObject),' ) ');
  With C^ do
    While Count>0 do AtFree(Count-1);
  Writeln ('Freed all objects. Memory available : ',Memavail);
  Writeln ('Lost : ',Initmem-Memavail,' bytes. ');
  Dispose(C,Done);
end.

```

TCollection.FreeItem

Synopsis: Destroy a non-nil item.

Declaration: `procedure FreeItem(Item: Pointer); Virtual`

Visibility: default

Description: `FreeItem` calls the destructor of `Item` if it is not nil.

Remark: This function is used internally by the `TCollection` object, and should not be called directly.

Errors: None.

See also: `TCollection.Free` (678), `TCollection.AtFree` (680)

TCollection.AtDelete

Synopsis: Delete item at certain position.

Declaration: `procedure AtDelete(Index: Sw_Integer)`

Visibility: default

Description: `AtDelete` deletes the pointer at position `Index` in the collection. It doesn't call the object's destructor.

Errors: If `Index` isn't valid then `Error` (683) is called with `CoIndexError`.

See also: `TCollection.Delete` (679)

Listing: `./objectex/ex33.pp`

Program `ex33`;

{ Program to demonstrate the TCollection.AtDelete method }

Uses `Objects, MyObject`; *{ For TMyObject definition and registration }*

Var `C : PCollection`;
 `M : PMyObject`;
 `I, InitMem : Longint`;

begin
 `Randomize`;

```

C:=New(PCollection, Init(120,10));
InitMem:=Memavail;
Writeln('Initial memory : ', InitMem);
For I:=1 to 100 do
begin
M:=New(PMyObject, Init);
M^.SetField(I-1);
C^.Insert(M);
end;
Writeln('Added 100 Items. Memory available : ', Memavail);
Write('Lost : ', InitMem-Memavail, ' bytes. ');
Write(' (Should be 100* ', SizeOf(TMyObject));
Writeln('=', 100*SizeOf(TMyObject), ') ');
With C^ do
While Count>0 do AtDelete(Count-1);
Writeln('Freed all objects. Memory available : ', Memavail);
Writeln('Lost : ', InitMem-Memavail, ' bytes. ');
Dispose(C, Done);
end.

```

TCollection.ForEach

Synopsis: Execute procedure for each item in the list.

Declaration: `procedure ForEach(Action: Pointer)`

Visibility: default

Description: `ForEach` calls `Action` for each element in the collection, and passes the element as an argument to `Action`.

`Action` is a procedural type variable that accepts a pointer as an argument.

Errors: None.

See also: `TCollection.FirstThat` ([675](#)), `TCollection.LastThat` ([674](#))

Listing: `./objectex/ex21.pp`

Program `ex21`;

{ Program to demonstrate the TCollection.ForEach method }

Uses `Objects, MyObject`; *{ For TMyObject definition and registration }*

Var `C` : `PCollection`;
`M` : `PMyObject`;
`I` : `Longint`;

Procedure `PrintField` (`Dummy`: `Pointer`; `P` : `PMyObject`);

begin
`Writeln` ('Field : ', `P^.GetField`);
end;

begin
`C:=New`(`PCollection`, `Init`(100,10));
For `I:=1 to 100 do`

```

    begin
    M:=New(PMyObject, Init);
    M^.SetField(100-I);
    C^.Insert(M);
    end;
    WriteLn ( 'Inserted ',C^.Count, ' objects ');
    C^.ForEach( @PrintField );
    C^.FreeAll;
    Dispose(C, Done);
end.

```

TCollection.SetLimit

Synopsis: Set maximum number of elements in the collection.

Declaration: `procedure SetLimit(ALimit: Sw_Integer); Virtual`

Visibility: default

Description: `SetLimit` sets the maximum number of elements in the collection. `ALimit` must not be less than `Count`, and should not be larger than `MaxCollectionSize`

For an example, see Pack (675).

Errors: None.

See also: `TCollection.Init` (670)

TCollection.Error

Synopsis: Set error code.

Declaration: `procedure Error(Code: Integer;Info: Integer); Virtual`

Visibility: default

Description: `Error` is called by the various `TCollection` methods in case of an error condition. The default behaviour is to make a call to `RunError` with an error of 212-Code.

This method can be overridden by descendent objects to implement a different error-handling.

Errors:

See also: `Abstract` (661)

TCollection.AtPut

Synopsis: Set collection item, overwriting an existing value.

Declaration: `procedure AtPut(Index: Sw_Integer;Item: Pointer)`

Visibility: default

Description: `AtPut` sets the element at position `Index` in the collection to `Item`. Any previous value is overwritten.

For an example, see Pack (675).

Errors: If `Index` isn't valid then `Error` (683) is called with `CoIndexError`.

TCollection.AtInsert

Synopsis: Insert an element at a certain position in the collection.

Declaration: `procedure AtInsert(Index: Sw_Integer;Item: Pointer)`

Visibility: default

Description: `AtInsert` inserts `Item` in the collection at position `Index`, shifting all elements by one position. In case the current limit is reached, the collection will try to expand with a call to `SetLimit`

Errors: If `Index` isn't valid then `Error (683)` is called with `CoIndexError`. If the collection fails to expand, then `coOverflow` is passed to `Error`.

See also: `TCollection.Insert (679)`

Listing: `./objectex/ex34.pp`

Program `ex34;`

{ Program to demonstrate the TCollection.AtInsert method }

Uses `Objects,MyObject; { For TMyObject definition and registration }`

Var `C : PCollection;`
`M : PMyObject;`
`I : Longint;`

Procedure `PrintField (Dummy: Pointer;P : PMyObject);`

begin
`WriteLn ('Field : ',P^.GetField);`
end;

begin
`Randomize;`
`C:=New(PCollection,Init(120,10));`
`WriteLn ('Inserting 100 records at random places.');`
`For I:=1 to 100 do`
`begin`
`M:=New(PMyObject,Init);`
`M^.SetField(I-1);`
`If I=1 then`
`C^.Insert(M)`
`else`
`With C^ do`
`AtInsert(Random(Count),M);`
`end;`
`WriteLn ('Values : ');`
`C^.Foreach(@PrintField);`
`Dispose(C,Done);`
end.

TCollection.Store

Synopsis: Write collection to a stream.

Declaration: `procedure Store(var S: TStream)`

Visibility: default

Description: Store writes the collection to the stream S. It does this by writing the current Count, Limit and Delta to the stream, and then writing each item to the stream.

The contents of the stream are then suitable for instantiating another collection with Load (671).

For an example, see TCollection.Load (671).

Errors: Errors returned are those by TStream.Put (717).

See also: TCollection.Load (671), TCollection.PutItem (685)

TCollection.PutItem

Synopsis: Put one item on the stream

Declaration: procedure PutItem(var S: TStream; Item: Pointer); Virtual

Visibility: default

Description: PutItem writes Item to stream S. This method is used internally by the TCollection object, and should not be called directly.

Errors: Errors are those returned by TStream.Put (717).

See also: TCollection.Store (684), TCollection.GetItem (673)

20.6 TDosStream

Description

TDosStream is a stream that stores its contents in a file. it overrides a couple of methods of TStream (712) for this.

In addition to the fields inherited from TStream (see TStream (712)), there are some extra fields, that describe the file. (mainly the name and the OS file handle)

No buffering in memory is done when using TDosStream. All data are written directly to the file. For a stream that buffers in memory, see TBufStream (666).

Method overview

Page	Method	Description
686	Close	Close the file.
686	Done	Closes the file and cleans up the instance.
686	Init	Instantiate a new instance of TDosStream.
688	Open	Open the file stream
689	Read	Read data from the stream to a buffer.
687	Seek	Set file position.
687	Truncate	Truncate the file on the current position.
689	Write	Write data from a buffer to the stream.

TDosStream.Init

Synopsis: Instantiate a new instance of TDosStream.

Declaration: constructor `Init(FileName: FNameStr; Mode: Word)`

Visibility: default

Description: `Init` instantiates an instance of `TDosStream`. The name of the file that contains (or will contain) the data of the stream is given in `FileName`. The `Mode` parameter determines whether a new file should be created and what access rights you have on the file. It can be one of the following constants:

stCreate Creates a new file.

stOpenRead Read access only.

stOpenWrite Write access only.

stOpenRead and write access.

For an example, see `TDosStream.Truncate` (687).

Errors: On error, `Status` (??) is set to `stInitError`, and `ErrorInfo` is set to the dos error code.

See also: `TDosStream.Done` (686)

TDosStream.Done

Synopsis: Closes the file and cleans up the instance.

Declaration: destructor `Done; Virtual`

Visibility: default

Description: `Done` closes the file if it was open and cleans up the instance of `TDosStream`.
for an example, see e.g. `TDosStream.Truncate` (687).

Errors: None.

See also: `TDosStream.Init` (686), `TDosStream.Close` (686)

TDosStream.Close

Synopsis: Close the file.

Declaration: procedure `Close; Virtual`

Visibility: default

Description: `Close` closes the file if it was open, and sets `Handle` to -1. Contrary to `Done` (686) it does not clean up the instance of `TDosStream`

For an example, see `TDosStream.Open` (688).

Errors: None.

See also: `TStream.Close` (716), `TDosStream.Init` (686), `TDosStream.Done` (686)

TDosStream.Truncate

Synopsis: Truncate the file on the current position.

Declaration: `procedure Truncate; Virtual`

Visibility: `default`

Description: If the status of the stream is `stOK`, then `Truncate` tries to truncate the stream size to the current file position.

Errors: If an error occurs, the stream's status is set to `stError` and `ErrorInfo` is set to the OS error code.

See also: `TStream.Truncate` (717), `TStream.GetSize` (714)

Listing: `./objectex/ex16.pp`

Program `ex16;`

{ Program to demonstrate the TStream.Truncate method }

Uses `Objects;`

Var `L : String;`
 `P : PString;`
 `S : PDosStream; { Only one with Truncate implemented. }`

begin

```

L:= 'Some constant string';
{ Buffer size of 100 }
S:=New(PDosStream, Init( 'test.dat', stcreate ));
Writeln ( 'Writing "', L, '" to stream with handle ', S^.Handle );
S^.WriteStr(@L);
S^.WriteStr(@L);
{ Close calls flush first }
S^.Close;
S^.Open ( stOpen );
Writeln ( 'Size of stream is : ', S^.GetSize );
P:=S^.ReadStr;
L:=P^;
DisposeStr(P);
Writeln ( 'Read "', L, '" from stream with handle ', S^.Handle );
S^.Truncate;
Writeln ( 'Truncated stream. Size is : ', S^.GetSize );
S^.Close;
Dispose ( S, Done );

```

end.

TDosStream.Seek

Synopsis: Set file position.

Declaration: `procedure Seek(Pos: LongInt); Virtual`

Visibility: `default`

Description: If the stream's status is `stOK`, then `Seek` sets the file position to `Pos`. `Pos` is a zero-based offset, counted from the beginning of the file.

Errors: In case an error occurs, the stream's status is set to `stSeekError`, and the OS error code is stored in `ErrorInfo`.

See also: `TStream.Seek` (718), `TStream.GetPos` (714)

Listing: `./objectex/ex17.pp`

Program `ex17`;

{ Program to demonstrate the TStream.Seek method }

Uses `Objects`;

Var `L : String`;
 `Marker : Word`;
 `P : PString`;
 `S : PDosStream`;

begin
 `L := 'Some constant string';`
 { Buffer size of 100 }
 `S := New(PDosStream, Init('test.dat', stcreate));`
 `WriteLn ('Writing ', L, ' to stream.');`
 `S^.WriteStr(@L);`
 `Marker := S^.GetPos;`
 `WriteLn ('Set marker at ', Marker);`
 `L := 'Some other constant String';`
 `WriteLn ('Writing ', L, ' to stream.');`
 `S^.WriteStr(@L);`
 `S^.Close;`
 `S^.Open (stOpenRead);`
 `WriteLn ('Size of stream is : ', S^.GetSize);`
 `WriteLn ('Seeking to marker');`
 `S^.Seek(Marker);`
 `P := S^.ReadStr;`
 `L := P^;`
 `DisposeStr(P);`
 `WriteLn ('Read ', L, ' from stream.');`
 `S^.Close;`
 `Dispose (S, Done);`
end.

TDosStream.Open

Synopsis: Open the file stream

Declaration: `procedure Open(OpenMode: Word); Virtual`

Visibility: default

Description: If the stream's status is `stOK`, and the stream is closed then `Open` re-opens the file stream with mode `OpenMode`. This call can be used after a `Close` (686) call.

Errors: If an error occurs when re-opening the file, then `Status` is set to `stOpenError`, and the OS error code is stored in `ErrorInfo`

See also: `TStream.Open` (716), `TDosStream.Close` (686)

Listing: ./objectex/ex14.pp

```

Program ex14;

{ Program to demonstrate the TStream.Close method }

Uses Objects;

Var L : String;
    P : PString;
    S : PDosStream; { Only one with Close implemented. }

begin
    L:= 'Some constant string';
    S:=New(PDosStream, Init('test.dat', stcreate));
    WriteIn ('Writing "', L, '" to stream with handle ', S^.Handle);
    S^.WriteStr(@L);
    S^.Close;
    WriteIn ('Closed stream. File handle is ', S^.Handle);
    S^.Open (stOpenRead);
    P:=S^.ReadStr;
    L:=P^;
    DisposeStr(P);
    WriteIn ('Read "', L, '" from stream with handle ', S^.Handle);
    S^.Close;
    Dispose (S, Done);
end.

```

TDosStream.Read

Synopsis: Read data from the stream to a buffer.

Declaration: `procedure Read(var Buf; Count: LongInt); Virtual`

Visibility: default

Description: If the Stream is open and the stream status is `stOK` then `Read` will read `Count` bytes from the stream and place them in `Buf`.

For an example, see `TStream.Read` ([719](#)).

Errors: In case of an error, `Status` is set to `StReadError`, and `ErrorInfo` gets the OS specific error, or 0 when an attempt was made to read beyond the end of the stream.

See also: `TStream.Read` ([719](#)), `TDosStream.Write` ([689](#))

TDosStream.Write

Synopsis: Write data from a buffer to the stream.

Declaration: `procedure Write(var Buf; Count: LongInt); Virtual`

Visibility: default

Description: If the Stream is open and the stream status is `stOK` then `Write` will write `Count` bytes from `Buf` and place them in the stream.

For an example, see `TStream.Read` ([719](#)).

Errors: In case of an error, `Status` is set to `StWriteError`, and `ErrorInfo` gets the OS specific error.

See also: `TStream.Write` (719), `TDosStream.Read` (689)

20.7 TMemoryStream

Description

The `TMemoryStream` object implements a stream that stores its data in memory. The data is stored on the heap, with the possibility to specify the maximum amount of data, and the size of the memory blocks being used.

Method overview

Page	Method	Description
690	<code>Done</code>	Clean up memory and destroy the object instance.
690	<code>Init</code>	Initialize memory stream, reserves memory for stream data.
691	<code>Read</code>	Read data from the stream to a location in memory.
691	<code>Truncate</code>	Set the stream size to the current position.
692	<code>Write</code>	Write data to the stream.

TMemoryStream.Init

Synopsis: Initialize memory stream, reserves memory for stream data.

Declaration: constructor `Init(ALimit: LongInt; ABlockSize: Word)`

Visibility: default

Description: `Init` instantiates a new `TMemoryStream` object. The `memorystreamobject` will initially allocate at least `ALimit` bytes memory, divided into memory blocks of size `ABlockSize`. The number of blocks needed to get to `ALimit` bytes is rounded up.

By default, the number of blocks is 1, and the size of a block is 8192. This is selected if you specify 0 as the blocksize.

For an example, see e.g. `TStream.CopyFrom` (720).

Errors: If the stream cannot allocate the initial memory needed for the memory blocks, then the stream's status is set to `stInitError`.

See also: `TMemoryStream.Done` (690)

TMemoryStream.Done

Synopsis: Clean up memory and destroy the object instance.

Declaration: destructor `Done; Virtual`

Visibility: default

Description: `Done` releases the memory blocks used by the stream, and then cleans up the memory used by the stream object itself.

For an example, see e.g. `TStream.CopyFrom` (720).

Errors: None.

See also: `TMemoryStream.Init` (690)

TMemoryStream.Truncate

Synopsis: Set the stream size to the current position.

Declaration: `procedure Truncate; Virtual`

Visibility: default

Description: `Truncate` sets the size of the memory stream equal to the current position. It de-allocates any memory-blocks that are no longer needed, so that the new size of the stream is the current position in the stream, rounded up to the first multiple of the stream blocksize.

Errors: If an error occurs during memory de-allocation, the stream's status is set to `stError`

See also: `TStream.Truncate` (717)

Listing: `./objectex/ex20.pp`

Program `ex20;`

{ Program to demonstrate the TMemoryStream.Truncate method }

Uses `Objects;`

Var `L : String;`
 `P : PString;`
 `S : PMemoryStream;`
 `I, InitMem : Longint;`

begin

```
  InitMem:=Memavail;
  L:='Some constant string';
  { Buffer size of 100 }
  S:=New(PMemoryStream, Init(1000,100));
  Writeln ('Free memory : ',Memavail);
  Writeln ('Writing 100 times "',L,'" to stream. ');
  For I:=1 to 100 do
    S^.WriteStr(@L);
  Writeln ('Finished. Free memory : ',Memavail);
  S^.Seek(100);
  S^.Truncate;
  Writeln ('Truncated at byte 100. Free memory : ',Memavail);
  Dispose (S,Done);
  Writeln ('Finished. Lost ',InitMem-Memavail,' Bytes. ');
```

end.

TMemoryStream.Read

Synopsis: Read data from the stream to a location in memory.

Declaration: `procedure Read(var Buf;Count: LongInt); Virtual`

Visibility: default

Description: `Read` reads `Count` bytes from the stream to `Buf`. It updates the position of the stream.

For an example, see `TStream.Read` (719).

Errors: If there is not enough data available, no data is read, and the stream's status is set to `stReadError`.

See also: `TStream.Read` (719), `TMemoryStream.Write` (692)

TMemoryStream.Write

Synopsis: Write data to the stream.

Declaration: `procedure Write(var Buf; Count: LongInt); Virtual`

Visibility: default

Description: Write copies Count bytes from Buf to the stream. It updates the position of the stream.

If not enough memory is available to hold the extra Count bytes, then the stream will try to expand, by allocating as much blocks with size BlkSize (as specified in the constructor call Init (690)) as needed.

For an example, see TStream.Read (719).

Errors: If the stream cannot allocate more memory, then the status is set to stWriteError

See also: TStream.Write (719), TMemoryStream.Read (691)

20.8 TObject**Description**

This type serves as the basic object for all other objects in the Objects unit.

Method overview

Page	Method	Description
693	Done	Destroy an object.
692	Free	Destroy an object and release all memory.
692	Init	Construct (initialize) a new object
693	Is_Object	

TObject.Init

Synopsis: Construct (initialize) a new object

Declaration: `constructor Init`

Visibility: default

Description: Instantiates a new object of type TObject. It fills the instance up with Zero bytes.

For an example, see Free (692)

Errors: None.

See also: TObject.Free (692), TObject.Done (693)

TObject.Free

Synopsis: Destroy an object and release all memory.

Declaration: `procedure Free`

Visibility: default

Description: `Free` calls the destructor of the object, and releases the memory occupied by the instance of the object.

Errors: No checking is performed to see whether `self` is `nil` and whether the object is indeed allocated on the heap.

See also: `TObject.Init` (692), `TObject.Done` (693)

Listing: `./objectex/ex7.pp`

```

program ex7;

  { Program to demonstrate the TObject.Free call }

Uses Objects;

Var O : PObject;

begin
  WriteLn ( 'Memavail : ', Memavail );
  // Allocate memory for object.
  O := New(PObject, Init);
  WriteLn ( 'Memavail : ', Memavail );
  // Free memory of object.
  O^.free;
  WriteLn ( 'Memavail : ', Memavail );
end.

```

TObject.Is_Object

Declaration: `function Is_Object(P: Pointer) : Boolean`

Visibility: default

TObject.Done

Synopsis: Destroy an object.

Declaration: `destructor Done; Virtual`

Visibility: default

Description: `Done`, the destructor of `TObject` does nothing. It is mainly intended to be used in the `TObject.Free` (692) method.

The destructore `Done` does not free the memory occupied by the object.

Errors: None.

See also: `TObject.Free` (692), `TObject.Init` (692)

Listing: `./objectex/ex8.pp`

```

program ex8;

  { Program to demonstrate the TObject.Done call }

Uses Objects;

```

```

Var O : PObject;

begin
  Writeln ( 'Memavail : ', Memavail );
  // Allocate memory for object.
  O:=New(PObject, Init);
  Writeln ( 'Memavail : ', Memavail );
  O^.Done;
  Writeln ( 'Memavail : ', Memavail );
end.

```

20.9 TPoint

Description

Record describing a point in a 2 dimensional plane.

20.10 TRect

Description

Describes a rectangular region in a plane.

Method overview

Page	Method	Description
699	Assign	Set rectangle corners.
696	Contains	Determine if a point is inside the rectangle
696	Copy	Copy cornerpoints from another rectangle.
694	Empty	Is the surface of the rectangle zero
695	Equals	Do the corners of the rectangles match
699	Grow	Expand rectangle with certain size.
697	Intersect	Reduce rectangle to intersection with another rectangle
698	Move	Move rectangle along a vector.
697	Union	Enlarges rectangle to encompass another rectangle.

TRect.Empty

Synopsis: Is the surface of the rectangle zero

Declaration: `function Empty : Boolean`

Visibility: default

Description: `Empty` returns `True` if the rectangle defined by the corner points A, B has zero or negative surface.

Errors: None.

See also: `TRect.Equals` ([695](#)), `TRect.Contains` ([696](#))

Listing: `./objectex/ex1.pp`

```

Program ex1;

{ Program to demonstrate TRect.Empty }

Uses objects;

Var ARect,BRect : TRect;
    P : TPoint;

begin
  With ARect.A do
    begin
      X:=10;
      Y:=10;
    end;
  With ARect.B do
    begin
      X:=20;
      Y:=20;
    end;
  { Offset B by (5,5) }
  With BRect.A do
    begin
      X:=15;
      Y:=15;
    end;
  With BRect.B do
    begin
      X:=25;
      Y:=25;
    end;
  { Point }
  With P do
    begin
      X:=15;
      Y:=15;
    end;
  WriteIn ( 'A empty : ',ARect.Empty);
  WriteIn ( 'B empty : ',BRect.Empty);
  WriteIn ( 'A Equals B : ',ARect.Equals(BRect));
  WriteIn ( 'A Contains (15,15) : ',ARect.Contains(P));
end.

```

TRect.Equals

Synopsis: Do the corners of the rectangles match

Declaration: `function Equals(R: TRect) : Boolean`

Visibility: default

Description: `Equals` returns `True` if the rectangle has the same corner points A,B as the rectangle R, and `False` otherwise.

For an example, see `TRect.Empty` ([694](#))

Errors: None.

See also: `TRect.Empty` ([694](#)), `TRect.Contains` ([696](#))

TRect.Contains

Synopsis: Determine if a point is inside the rectangle

Declaration: `function Contains(P: TPoint) : Boolean`

Visibility: default

Description: `Contains` returns `True` if the point `P` is contained in the rectangle (including borders), `False` otherwise.

Errors: None.

See also: `TRect.Intersect` ([697](#)), `TRect.Equals` ([695](#))

TRect.Copy

Synopsis: Copy cornerpoints from another rectangle.

Declaration: `procedure Copy(R: TRect)`

Visibility: default

Description: Assigns the rectangle `R` to the object. After the call to `Copy`, the rectangle `R` has been copied to the object that invoked `Copy`.

Errors: None.

See also: `TRect.Assign` ([699](#))

Listing: `./objectex/ex2.pp`

```
Program ex2;

{ Program to demonstrate TRect.Copy }

Uses objects;

Var ARect, BRect, CRect : TRect;

begin
  ARect.Assign(10,10,20,20);
  BRect.Assign(15,15,25,25);
  CRect.Copy(ARect);
  If ARect.Equals(CRect) Then
    Writeln ( 'ARect equals CRect' )
  Else
    Writeln ( 'ARect does not equal CRect !' );
end.
```

TRect.Union

Synopsis: Enlarges rectangle to encompass another rectangle.

Declaration: `procedure Union(R: TRect)`

Visibility: default

Description: `Union` enlarges the current rectangle so that it becomes the union of the current rectangle with the rectangle `R`.

Errors: None.

See also: `TRect.Intersect` ([697](#))

Listing: `./objectex/ex3.pp`

Program `ex3`;

{ Program to demonstrate TRect.Union }

Uses `objects`;

Var `ARect, BRect, CRect : TRect`;

begin

`ARect.Assign(10,10,20,20);`

`BRect.Assign(15,15,25,25);`

{ CRect is union of ARect and BRect }

`CRect.Assign(10,10,25,25);`

{ Calculate it explicitly }

`ARect.Union(BRect);`

If `ARect.Equals(CRect)` **Then**

`Writeln ('ARect equals CRect')`

Else

`Writeln ('ARect does not equal CRect !');`

end.

TRect.Intersect

Synopsis: Reduce rectangle to intersection with another rectangle

Declaration: `procedure Intersect(R: TRect)`

Visibility: default

Description: `Intersect` makes the intersection of the current rectangle with `R`. If the intersection is empty, then the rectangle is set to the empty rectangle at coordinate (0,0).

Errors: None.

See also: `TRect.Union` ([697](#))

Listing: `./objectex/ex4.pp`

```

Program ex4;

{ Program to demonstrate TRect.Intersect }

Uses objects;

Var ARect,BRect,CRect : TRect;

begin
  ARect.Assign(10,10,20,20);
  BRect.Assign(15,15,25,25);
  { CRect is intersection of ARect and BRect }
  CRect.Assign(15,15,20,20);
  { Calculate it explicitly }
  ARect.Intersect(BRect);
  If ARect.Equals(CRect) Then
    Writeln ( 'ARect equals CRect' )
  Else
    Writeln ( 'ARect does not equal CRect !' );
  BRect.Assign(25,25,30,30);
  ARect.Intersect(BRect);
  If ARect.Empty Then
    Writeln ( 'ARect is empty' );
end.

```

TRect.Move

Synopsis: Move rectangle along a vector.

Declaration: `procedure Move(ADX: Sw_Integer;ADY: Sw_Integer)`

Visibility: default

Description: Move moves the current rectangle along a vector with components (ADX , ADY). It adds ADX to the X-coordinate of both corner points, and ADY to both end points.

Errors: None.

See also: TRect.Grow ([699](#))

Listing: ./objectex/ex5.pp

```

Program ex5;

{ Program to demonstrate TRect.Move }

Uses objects;

Var ARect,BRect : TRect;

begin
  ARect.Assign(10,10,20,20);
  ARect.Move(5,5);
  // Brect should be where new ARect is.
  BRect.Assign(15,15,25,25);

```

```

If ARect.Equals(BRect) Then
  WriteLn ( 'ARect equals BRect' )
Else
  WriteLn ( 'ARect does not equal BRect !' );
end.

```

TRect.Grow

Synopsis: Expand rectangle with certain size.

Declaration: `procedure Grow(ADX: Sw_Integer;ADY: Sw_Integer)`

Visibility: default

Description: `Grow` expands the rectangle with an amount `ADX` in the X direction (both on the left and right side of the rectangle, thus adding a length $2*ADX$ to the width of the rectangle), and an amount `ADY` in the Y direction (both on the top and the bottom side of the rectangle, adding a length $2*ADY$ to the height of the rectangle).

`ADX` and `ADY` can be negative. If the resulting rectangle is empty, it is set to the empty rectangle at `(0,0)`.

Errors: None.

See also: `TRect.Move` ([698](#))

Listing: `./objectex/ex6.pp`

```

Program ex6;

{ Program to demonstrate TRect.Grow }

Uses objects;

Var ARect,BRect : TRect;

begin
  ARect.Assign(10,10,20,20);
  ARect.Grow(5,5);
  // Brect should be where new ARect is.
  BRect.Assign(5,5,25,25);
  If ARect.Equals(BRect) Then
    WriteLn ( 'ARect equals BRect' )
  Else
    WriteLn ( 'ARect does not equal BRect !' );
end.

```

TRect.Assign

Synopsis: Set rectangle corners.

Declaration: `procedure Assign(XA: Sw_Integer;YA: Sw_Integer;XB: Sw_Integer;YB: Sw_Integer)`

Visibility: default

Description: Assign sets the corner points of the rectangle to (XA, YA) and (Xb, Yb).

For an example, see TRect.Copy ([696](#)).

Errors: None.

See also: TRect.Copy ([696](#))

20.11 TResourceCollection

Description

A TResourceCollection manages a collection of resource names. It stores the position and the size of a resource, as well as the name of the resource. It stores these items in records that look like this:

```
TYPE
  TResourceItem = packed RECORD
    Posn: LongInt;
    Size: LongInt;
    Key : String;
  End;
  PResourceItem = ^TResourceItem;
```

It overrides some methods of TStringCollection in order to accomplish this.

Remark: Remark that the TResourceCollection manages the names of the resources and their associated positions and sizes, it doesn't manage the resources themselves.

Method overview

Page	Method	Description
701	FreeItem	Release memory occupied by item.
701	GetItem	Read an item from the stream.
700	KeyOf	Return the key of an item in the collection.
701	PutItem	Write an item to the stream.

TResourceCollection.KeyOf

Synopsis: Return the key of an item in the collection.

Declaration: function KeyOf(Item: Pointer) : Pointer; Virtual

Visibility: default

Description: KeyOf returns the key of an item in the collection. For resources, the key is a pointer to the string with the resource name.

Errors: None.

See also: TStringCollection.Compare ([721](#))

TResourceCollection.GetItem

Synopsis: Read an item from the stream.

Declaration: `function GetItem(var S: TStream) : Pointer; Virtual`

Visibility: default

Description: `GetItem` reads a resource item from the stream `S`. It reads the position, size and name from the stream, in that order. It DOES NOT read the resource itself from the stream.

The resulting item is not inserted in the collection. This call is mainly for internal use by the `TCollection.Load (671)` method.

Errors: Errors returned are those by `TStream.Read (719)`

See also: `TCollection.Load (671)`, `TStream.Read (719)`

TResourceCollection.FreeItem

Synopsis: Release memory occupied by item.

Declaration: `procedure FreeItem(Item: Pointer); Virtual`

Visibility: default

Description: `FreeItem` releases the memory occupied by `Item`. It de-allocates the name, and then the resource item record.

It does NOT remove the item from the collection.

Errors: None.

See also: `TCollection.FreeItem (681)`

TResourceCollection.PutItem

Synopsis: Write an item to the stream.

Declaration: `procedure PutItem(var S: TStream; Item: Pointer); Virtual`

Visibility: default

Description: `PutItem` writes `Item` to the stream `S`. It does this by writing the position and size and name of the resource item to the stream.

This method is used primarily by the `Store (684)` method.

Errors: Errors returned are those by `TStream.Write (719)`.

See also: `TCollection.Store (684)`

20.12 TResourceFile**Description**

`TResourceFile (701)` represents the resources in a binary file image.

Method overview

Page	Method	Description
702	Count	Number of resources in the file
704	Delete	Delete a resource from the file
702	Done	Destroy the instance and remove it from memory.
703	Flush	Writes the resources to the stream.
703	Get	Return a resource by key name.
702	Init	Instantiate a new instance.
703	KeyAt	Return the key of the item at a certain position.
704	Put	Set a resource by key name.
703	SwitchTo	Write resources to a new stream.

TResourceFile.Init

Synopsis: Instantiate a new instance.

Declaration: constructor `Init(AStream: PStream)`

Visibility: default

Description: `Init` instantiates a new instance of a `TResourceFile` object. If `AStream` is not nil then it is considered as a stream describing an executable image on disk.

`Init` will try to position the stream on the start of the resources section, and read all resources from the stream.

Errors: None.

See also: `TResourceFile.Done` ([702](#))

TResourceFile.Done

Synopsis: Destroy the instance and remove it from memory.

Declaration: destructor `Done; Virtual`

Visibility: default

Description: `Done` cleans up the instance of the `TResourceFile` Object. If `Stream` was specified at initialization, then `Stream` is disposed of too.

Errors: None.

See also: `TResourceFile.Init` ([702](#))

TResourceFile.Count

Synopsis: Number of resources in the file

Declaration: function `Count : Sw_Integer`

Visibility: default

Description: `Count` returns the number of resources. If no resources were read, zero is returned.

Errors: None.

See also: `TResourceFile.Init` ([702](#))

TResourceFile.KeyAt

Synopsis: Return the key of the item at a certain position.

Declaration: `function KeyAt(I: Sw_Integer) : String`

Visibility: default

Description: `KeyAt` returns the key (the name) of the `I`-th resource.

Errors: In case `I` is invalid, `TCollection.Error` will be executed.

See also: `TResourceFile.Get` (703)

TResourceFile.Get

Synopsis: Return a resource by key name.

Declaration: `function Get(Key: String) : PObject`

Visibility: default

Description: `Get` returns a pointer to a instance of a resource identified by `Key`. If `Key` cannot be found in the list of resources, then `Nil` is returned.

Errors: Errors returned may be those by `TStream.Get`

TResourceFile.SwitchTo

Synopsis: Write resources to a new stream.

Declaration: `function SwitchTo(AStream: PStream; Pack: Boolean) : PStream`

Visibility: default

Description: `SwitchTo` switches to a new stream to hold the resources in. `AStream` will be the new stream after the call to `SwitchTo`.

If `Pack` is true, then all the known resources will be copied from the current stream to the new stream (`AStream`). If `Pack` is False, then only the current resource is copied.

The return value is the value of the original stream: `Stream`.

The `Modified` flag is set as a consequence of this call.

Errors: Errors returned can be those of `TStream.Read` (719) and `TStream.Write` (719).

See also: `TResourceFile.Flush` (703)

TResourceFile.Flush

Synopsis: Writes the resources to the stream.

Declaration: `procedure Flush`

Visibility: default

Description: If the `Modified` flag is set to `True`, then `Flush` writes the resources to the stream `Stream`. It sets the `Modified` flag to true after that.

Errors: Errors can be those by `TStream.Seek` (718) and `TStream.Write` (719).

See also: `TResourceFile.SwitchTo` (703)

TResourceFile.Delete

Synopsis: Delete a resource from the file

Declaration: `procedure Delete(Key: String)`

Visibility: `default`

Description: `Delete` deletes the resource identified by `Key` from the collection. It sets the `Modified` flag to `true`.

Errors: `None`.

See also: `TResourceFile.Flush` ([703](#))

TResourceFile.Put

Synopsis: Set a resource by key name.

Declaration: `procedure Put(Item: PObject;Key: String)`

Visibility: `default`

Description: `Put` sets the resource identified by `Key` to `Item`. If no such resource exists, a new one is created. The item is written to the stream.

Errors: Errors returned may be those by `TStream.Put` ([717](#)) and `TStream.Seek`

See also: `TResourceFile.Get` ([703](#))

20.13 TSortedCollection**Description**

`TSortedCollection` is an abstract class, implementing a sorted collection. You should never use an instance of `TSortedCollection` directly, instead you should declare a descendent type, and override the `Compare` ([706](#)) method.

Because the collection is ordered, `TSortedCollection` overrides some `TCollection` methods, to provide faster routines for lookup.

The `Compare` ([706](#)) method decides how elements in the collection should be ordered. Since `TCollection` has no way of knowing how to order pointers, you must override the compare method.

Additionally, `TCollection` provides a means to filter out duplicates. if you set `Duplicates` to `False` (the default) then duplicates will not be allowed.

The example below defines a descendent of `TSortedCollection` which is used in the examples.

Method overview

Page	Method	Description
706	Compare	Compare two items in the collection.
706	IndexOf	Return index of an item in the collection.
705	Init	Instantiates a new instance of a TSortedCollection
708	Insert	Insert new item in collection.
705	KeyOf	Return the key of an item
705	Load	Instantiates a new instance of a TSortedCollection and loads it from stream.
707	Search	Search for item with given key.
709	Store	Write the collection to the stream.

TSortedCollection.Init

Synopsis: Instantiates a new instance of a TSortedCollection

Declaration: constructor Init(ALimit: Sw_Integer;ADelta: Sw_Integer)

Visibility: default

Description: Init calls the inherited constructor (see TCollection.Init ([670](#))) and sets the Duplicates flag to false.

You should not call this method directly, since TSortedCollection is a abstract class. Instead, the descendent classes should call it via the inherited keyword.

Errors: None.

See also: TSortedCollection.Load ([705](#)), TCollection.Done ([672](#))

TSortedCollection.Load

Synopsis: Instantiates a new instance of a TSortedCollection and loads it from stream.

Declaration: constructor Load(var S: TStream)

Visibility: default

Description: Load calls the inherited constructor (see TCollection.Load ([671](#))) and reads the Duplicates flag from the stream..

You should not call this method directly, since TSortedCollection is a abstract class. Instead, the descendent classes should call it via the inherited keyword.

For an example, see TCollection.Load ([671](#)).

Errors: None.

See also: TSortedCollection.Init ([705](#)), TCollection.Done ([672](#))

TSortedCollection.KeyOf

Synopsis: Return the key of an item

Declaration: function KeyOf(Item: Pointer) : Pointer; Virtual

Visibility: default

Description: `KeyOf` returns the key associated with `Item`. `TSortedCollection` returns the item itself as the key, descendent objects can override this method to calculate a (unique) key based on the item passed (such as hash values).

Keys are used to sort the objects, they are used to search and sort the items in the collection. If descendent types override this method then it allows possibly for faster search/sort methods based on keys rather than on the objects themselves.

Errors: None.

See also: `TSortedCollection.IndexOf` (706), `TSortedCollection.Compare` (706)

TSortedCollection.IndexOf

Synopsis: Return index of an item in the collection.

Declaration: `function IndexOf(Item: Pointer) : Sw_Integer; Virtual`

Visibility: default

Description: `IndexOf` returns the index of `Item` in the collection. It searches for the object based on it's key. If duplicates are allowed, then it returns the index of last object that matches `Item`.

In case `Item` is not found in the collection, -1 is returned.

For an example, see `TCollection.IndexOf` (673)

Errors: None.

See also: `TSortedCollection.Search` (707), `TSortedCollection.Compare` (706)

TSortedCollection.Compare

Synopsis: Compare two items in the collection.

Declaration: `function Compare(Key1: Pointer;Key2: Pointer) : Sw_Integer; Virtual`

Visibility: default

Description: `Compare` is an abstract method that should be overridden by descendent objects in order to compare two items in the collection. This method is used in the `Search` (707) method and in the `Insert` (708) method to determine the ordering of the objects.

The function should compare the two keys of items and return the following function results:

Result < 0 If `Key1` is logically before `Key2` (`Key1<Key2`)

Result = 0 If `Key1` and `Key2` are equal. (`Key1=Key2`)

Result > 0 If `Key1` is logically after `Key2` (`Key1>Key2`)

Errors: An 'abstract run-time error' will be generated if you call `TSortedCollection.Compare` directly.

See also: `TSortedCollection.IndexOf` (706), `TSortedCollection.Search` (707)

Listing: `./objectex/mysortc.pp`

Unit MySortC;

Interface

Uses Objects;

Type

```

PMySortedCollection = ^TMySortedCollection;
TMySortedCollection = Object(TSortedCollection)
    Function Compare (Key1,Key2 : Pointer): Sw_integer; virtual;
    end;

```

Implementation

Uses MyObject;

Function TMySortedCollection.Compare (Key1,Key2 : Pointer) : sw_integer;

begin

```

    Compare:=PMyobject(Key1)^.GetField - PMyObject(Key2)^.GetField;

```

end;

end.

TSortedCollection.Search

Synopsis: Search for item with given key.

Declaration: `function Search(Key: Pointer;var Index: Sw_Integer) : Boolean; Virtual`

Visibility: default

Description: Search looks for the item with key Key and returns the position of the item (if present) in the collection in Index.

Instead of a linear search as TCollection does, TSortedCollection uses a binary search based on the keys of the objects. It uses the Compare (706) function to implement this search.

If the item is found, Search returns True, otherwise False is returned.

Errors: None.

See also: TCollection.IndexOf (673)

Listing: ./objectex/ex36.pp

Program ex36;

{ Program to demonstrate the TSortedCollection.Insert method }

Uses Objects, MyObject, MySortC;

{ For TMyObject, TMySortedCollection definition and registration }

Var C : PSortedCollection;

 M : PMyObject;

 I : Longint;

Procedure PrintField (Dummy: Pointer;P : PMyObject);


```

begin
  Writeln ( 'Field : ', P^.GetField);
end;

begin
  Randomize;
  C:=New( PMySortedCollection , Init (120,10));
  C^.Duplicates:=True;
  Writeln ( 'Inserting 100 records at random places. ');
  For I:=1 to 100 do
    begin
      M:=New(PMyObject, Init);
      M^.SetField(Random(100));
      C^.Insert(M)
    end;
  M:=New(PMyObject, Init);
  Repeat;
    Write ( 'Value to search for (-1 stops) : ');
    read (I);
    If I<>-1 then
      begin
        M^.SetField(i);
        If Not C^.Search (M,I) then
          Writeln ( 'No such value found')
        else
          begin
            Write ( 'Value ', PMyObject(C^.At(I))^ .GetField);
            Writeln ( ' present at position ', I);
          end;
        end;
      Until I=-1;
      Dispose (M, Done);
      Dispose (C, Done);
    end.

```

TSortedCollection.Insert

Synopsis: Insert new item in collection.

Declaration: `procedure Insert(Item: Pointer); Virtual`

Visibility: default

Description: `Insert` inserts an item in the collection at the correct position, such that the collection is ordered at all times. You should never use `Atinsert` (684), since then the collection ordering is not guaranteed.

If `Item` is already present in the collection, and `Duplicates` is `False`, the item will not be inserted.

Errors: None.

See also: `TCollection.AtInsert` (684)

Listing: `./objectex/ex35.pp`

```

Program ex35;

{ Program to demonstrate the TSortedCollection.Insert method }

Uses Objects, MyObject, MySortC;
      { For TMyObject, TMySortedCollection definition and registration }

Var C : PSortedCollection;
      M : PMyObject;
      I : Longint;

Procedure PrintField (Dummy: Pointer; P : PMyObject);

begin
  WriteLn ( 'Field : ', P^.GetField );
end;

begin
  Randomize;
  C:=New( PMySortedCollection, Init(120,10));
  WriteLn ( 'Inserting 100 records at random places.' );
  For I:=1 to 100 do
    begin
      M:=New( PMyObject, Init );
      M^.SetField( Random(100));
      C^.Insert( M )
    end;
    WriteLn ( 'Values : ' );
    C^.Foreach( @PrintField );
    Dispose(C, Done);
end.

```

TSortedCollection.Store

Synopsis: Write the collection to the stream.

Declaration: `procedure Store(var S: TStream)`

Visibility: default

Description: Store writes the collection to the stream S. It does this by calling the inherited TCollection.Store (684), and then writing the Duplicates flag to the stream.

After a Store, the collection can be loaded from the stream with the constructor Load (705)

For an example, see TCollection.Load (671).

Errors: Errors can be those of TStream.Put (717).

See also: TSortedCollection.Load (705)

20.14 TStrCollection

Description

The `TStrCollection` object manages a sorted collection of null-terminated strings (pchar strings). To this end, it overrides the `Compare` (706) method of `TSortedCollection`, and it introduces methods to read/write strings from a stream.

Method overview

Page	Method	Description
710	<code>Compare</code>	Compare two strings in the collection.
711	<code>FreeItem</code>	Free null-terminated string from the collection.
711	<code>GetItem</code>	Read a null-terminated string from the stream.
711	<code>PutItem</code>	Write a null-terminated string to the stream.

TStrCollection.Compare

Synopsis: Compare two strings in the collection.

Declaration: `function Compare(Key1: Pointer;Key2: Pointer) : Sw_Integer; Virtual`

Visibility: default

Description: `TStrCollection` overrides the `Compare` function so it compares the two keys as if they were pointers to strings. The compare is done case sensitive. It returns

-1 if the first string is alphabetically earlier than the second string.

0 if the two strings are equal.

1 if the first string is alphabetically later than the second string.

Errors: None.

See also: `TSortedCollection.Compare` (706)

Listing: `./objectex/ex38.pp`

Program `ex38`;

{ Program to demonstrate the TStrCollection.Compare method }

Uses `Objects , Strings`;

Var `C : PStrCollection`;
 `S : String`;
 `I : longint`;
 `P : Pchar`;

begin

`Randomize`;

`C:=New(PStrCollection , Init(120,10));`

`C^.Duplicates:=True; { Duplicates allowed }`

`WriteLn ('Inserting 100 records at random places.');`

For `I:=1 to 100 do`

begin

`Str(Random(100),S);`

```

    S:= 'String with value '+S;
    P:= StrAlloc (Length(S)+1);
    C^.Insert (StrPCopy(P,S));
    end;
  For I:=0 to 98 do
    With C^ do
      If Compare (At(I),At(I+1))=0 then
        WriteLn ('Duplicate string found at position ',I);
      Dispose(C,Done);
    end.

```

TStrCollection.GetItem

Synopsis: Read a null-terminated string from the stream.

Declaration: `function GetItem(var S: TStream) : Pointer; Virtual`

Visibility: default

Description: `GetItem` reads a null-terminated string from the stream `S` and returns a pointer to it. It doesn't insert the string in the collection.

This method is primarily introduced to be able to load and store the collection from and to a stream.

Errors: The errors returned are those of `TStream.StrRead` ([713](#)).

See also: `TStrCollection.PutItem` ([711](#))

TStrCollection.FreeItem

Synopsis: Free null-terminated string from the collection.

Declaration: `procedure FreeItem(Item: Pointer); Virtual`

Visibility: default

Description: `TStrCollection` overrides `FreeItem` so that the string pointed to by `Item` is disposed from memory.

Errors: None.

See also: `TCollection.FreeItem` ([681](#))

TStrCollection.PutItem

Synopsis: Write a null-terminated string to the stream.

Declaration: `procedure PutItem(var S: TStream; Item: Pointer); Virtual`

Visibility: default

Description: `PutItem` writes the string pointed to by `Item` to the stream `S`.

This method is primarily used in the `Load` and `Store` methods, and should not be used directly.

Errors: Errors are those of `TStream.StrWrite` ([718](#)).

See also: `TStrCollection.GetItem` ([711](#))

20.15 TStream

Description

The `TStream` object is the ancestor for all streaming objects, i.e. objects that have the capability to store and retrieve data.

It defines a number of methods that are common to all objects that implement streaming, many of them are virtual, and are only implemented in the descendent types.

Programs should not instantiate objects of type `TStream` directly, but instead instantiate a descendant type, such as `TDosStream`, `TMemoryStream`.

Method overview

Page	Method	Description
716	<code>Close</code>	Close the stream
720	<code>CopyFrom</code>	Copy data from another stream.
718	<code>Error</code>	Set stream status
717	<code>Flush</code>	Flush the stream data from the buffer, if any.
712	<code>Get</code>	Read an object definition from the stream.
714	<code>GetPos</code>	Return current position in the stream
714	<code>GetSize</code>	Return the size of the stream.
712	<code>Init</code>	
716	<code>Open</code>	Open the stream
717	<code>Put</code>	Write an object to the stream.
719	<code>Read</code>	Read data from stream to buffer.
715	<code>ReadStr</code>	Read a shortstring from the stream.
716	<code>Reset</code>	Reset the stream
718	<code>Seek</code>	Set stream position.
713	<code>StrRead</code>	Read a null-terminated string from the stream.
718	<code>StrWrite</code>	Write a null-terminated string to the stream.
717	<code>Truncate</code>	Truncate the stream size on current position.
719	<code>Write</code>	Write a number of bytes to the stream.
718	<code>WriteStr</code>	Write a pascal string to the stream.

TStream.Init

Declaration: constructor `Init`

Visibility: default

TStream.Get

Synopsis: Read an object definition from the stream.

Declaration: function `Get` : `PObject`

Visibility: default

Description: `Get` reads an object definition from a stream, and returns a pointer to an instance of this object.

Errors: On error, `TStream.Status` (??) is set, and `NIL` is returned.

See also: `TStream.Put` ([717](#))

Listing: ./objectex/ex9.pp

```

Program ex9;

{ Program to demonstrate TStream.Get and TStream.Put }

Uses Objects, MyObject; { Definition and registration of TMyObject }

Var Obj : PMyObject;
    S : PStream;

begin
  Obj:=New(PMyObject, Init);
  Obj^.SetField($1111);
  WriteLn ( 'Field value : ', Obj^.GetField);
  { Since Stream is an abstract type, we instantiate a TMemoryStream }
  S:=New(PMemoryStream, Init(100,10));
  S^.Put(Obj);
  WriteLn ( 'Disposing object');
  S^.Seek(0);
  Dispose(Obj, Done);
  WriteLn ( 'Reading object');
  Obj:=PMyObject(S^.Get);
  WriteLn ( 'Field Value : ', Obj^.GetField);
  Dispose(Obj, Done);
end.

```

TStream.StrRead

Synopsis: Read a null-terminated string from the stream.

Declaration: function StrRead : PChar

Visibility: default

Description: StrRead reads a string from the stream, allocates memory for it, and returns a pointer to a null-terminated copy of the string on the heap.

Errors: On error, Nil is returned.

See also: TStream.StrWrite (718), TStream.ReadStr (715)

Listing: ./objectex/ex10.pp

```

Program ex10;

{
  Program to demonstrate the TStream.StrRead TStream.StrWrite functions
}

Uses objects;

Var P : PChar;
    S : PStream;

begin
  P:= 'Constant Pchar string';
  WriteLn ( 'Writing to stream : "', P, '"');

```

```

S:=New(PMemoryStream, Init(100,10));
S^.StrWrite(P);
S^.Seek(0);
P:=Nil;
P:=S^.StrRead;
Dispose(S,Done);
Writeln('Read from stream : ',P,'');
Freemem(P, Strlen(P)+1);
end.

```

TStream.GetPos

Synopsis: Return current position in the stream

Declaration: `function GetPos : LongInt; Virtual`

Visibility: default

Description: If the stream's status is `stOk`, `GetPos` returns the current position in the stream. Otherwise it returns `-1`

Errors: `-1` is returned if the status is an error condition.

See also: `TStream.Seek` ([718](#)), `TStream.GetSize` ([714](#))

Listing: `./objectex/ex11.pp`

```

Program ex11;

{ Program to demonstrate the TStream.GetPos function }

Uses objects;

Var L : String;
    S : PStream;

begin
  L:= 'Some kind of string';
  S:=New(PMemoryStream, Init(100,10));
  Writeln('Stream position before write : ',S^.GetPos);
  S^.WriteStr(@L);
  Writeln('Stream position after write : ',S^.GetPos);
  Dispose(S,Done);
end.

```

TStream.GetSize

Synopsis: Return the size of the stream.

Declaration: `function GetSize : LongInt; Virtual`

Visibility: default

Description: If the stream's status is `stOk` then `GetSize` returns the size of the stream, otherwise it returns `-1`.

Errors: `-1` is returned if the status is an error condition.

See also: TStream.Seek ([718](#)), TStream.GetPos ([714](#))

Listing: ./objectex/ex12.pp

```

Program ex12;

{ Program to demonstrate the TStream.GetSize function }

Uses objects;

Var L : String;
    S : PStream;

begin
  L:= 'Some kind of string';
  S:=New(PMemoryStream, Init(100,10));
  WriteIn ( 'Stream size before write: ', S^.GetSize );
  S^.WriteStr(@L);
  WriteIn ( 'Stream size after write : ', S^.GetSize );
  Dispose(S, Done);
end.

```

TStream.ReadStr

Synopsis: Read a shortstring from the stream.

Declaration: function ReadStr : PString

Visibility: default

Description: ReadStr reads a string from the stream, copies it to the heap and returns a pointer to this copy. The string is saved as a pascal string, and hence is NOT null terminated.

Errors: On error (e.g. not enough memory), Nil is returned.

See also: TStream.StrRead ([713](#))

Listing: ./objectex/ex13.pp

```

Program ex13;

{
  Program to demonstrate the TStream.ReadStr TStream.WriteStr functions
}

Uses objects;

Var P : PString;
    L : String;
    S : PStream;

begin
  L:= 'Constant string line';
  WriteIn ( 'Writing to stream : "', L, '"' );
  S:=New(PMemoryStream, Init(100,10));
  S^.WriteStr(@L);
  S^.Seek(0);
  P:=S^.ReadStr;

```

```

L:=P^;
DisposeStr(P);
DisPose (S,Done);
WriteIn ('Read from stream : " ',L,'"');
end.

```

TStream.Open

Synopsis: Open the stream

Declaration: `procedure Open(OpenMode: Word); Virtual`

Visibility: default

Description: `Open` is an abstract method, that should be overridden by descendent objects. Since opening a stream depends on the stream's type this is not surprising.

For an example, see `TDosStream.Open` ([688](#)).

Errors: None.

See also: `TStream.Close` ([716](#)), `TStream.Reset` ([716](#))

TStream.Close

Synopsis: Close the stream

Declaration: `procedure Close; Virtual`

Visibility: default

Description: `Close` is an abstract method, that should be overridden by descendent objects. Since Closing a stream depends on the stream's type this is not surprising.

for an example, see `TDosStream.Open` ([688](#)).

Errors: None.

See also: `TStream.Open` ([716](#)), `TStream.Reset` ([716](#))

TStream.Reset

Synopsis: Reset the stream

Declaration: `procedure Reset`

Visibility: default

Description: `Reset` sets the stream's status to 0, as well as the `ErrorInfo`

Errors: None.

See also: `TStream.Open` ([716](#)), `TStream.Close` ([716](#))

TStream.Flush

Synopsis: Flush the stream data from the buffer, if any.

Declaration: `procedure Flush; Virtual`

Visibility: default

Description: `Flush` is an abstract method that should be overridden by descendent objects. It serves to enable the programmer to tell streams that implement a buffer to clear the buffer.

for an example, see `TBufStream.Flush` ([667](#)).

Errors: None.

See also: `TStream.Truncate` ([717](#))

TStream.Truncate

Synopsis: Truncate the stream size on current position.

Declaration: `procedure Truncate; Virtual`

Visibility: default

Description: `Truncate` is an abstract procedure that should be overridden by descendent objects. It serves to enable the programmer to truncate the size of the stream to the current file position.

For an example, see `TDosStream.Truncate` ([687](#)).

Errors: None.

See also: `TStream.Seek` ([718](#))

TStream.Put

Synopsis: Write an object to the stream.

Declaration: `procedure Put(P: PObject)`

Visibility: default

Description: `Put` writes the object pointed to by `P`. `P` should be non-nil. The object type must have been registered with `RegisterType` ([664](#)).

After the object has been written, it can be read again with `Get` ([712](#)).

For an example, see `TStream.Get` ([712](#));

Errors: No check is done whether `P` is `Nil` or not. Passing `Nil` will cause a run-time error 216 to be generated. If the object has not been registered, the status of the stream will be set to `stPutError`.

See also: `TStream.Get` ([712](#))

TStream.StrWrite

Synopsis: Write a null-terminated string to the stream.

Declaration: `procedure StrWrite(P: PChar)`

Visibility: default

Description: `StrWrite` writes the null-terminated string `P` to the stream. `P` can only be 65355 bytes long.

For an example, see `TStream.StrRead` (713).

Errors: None.

See also: `TStream.WriteString` (718), `TStream.StrRead` (713), `TStream.ReadStr` (715)

TStream.WriteString

Synopsis: Write a pascal string to the stream.

Declaration: `procedure WriteStr(P: PString)`

Visibility: default

Description: `StrWrite` writes the pascal string pointed to by `P` to the stream.

For an example, see `TStream.ReadStr` (715).

Errors: None.

See also: `TStream.StrWrite` (718), `TStream.StrRead` (713), `TStream.ReadStr` (715)

TStream.Seek

Synopsis: Set stream position.

Declaration: `procedure Seek(Pos: LongInt); Virtual`

Visibility: default

Description: `Seek` sets the position to `Pos`. This position is counted from the beginning, and is zero based. (i.e. `seek(0)` sets the position pointer on the first byte of the stream)

For an example, see `TDosStream.Seek` (687).

Errors: If `Pos` is larger than the stream size, `Status` is set to `StSeekError`.

See also: `TStream.GetPos` (714), `TStream.GetSize` (714)

TStream.Error

Synopsis: Set stream status

Declaration: `procedure Error(Code: Integer; Info: Integer); Virtual`

Visibility: default

Description: `Error` sets the stream's status to `Code` and `ErrorInfo` to `Info`. If the `StreamError` procedural variable is set, `Error` executes it, passing `Self` as an argument.

This method should not be called directly from a program. It is intended to be used in descendent objects.

Errors: None.

TStream.Read

Synopsis: Read data from stream to buffer.

Declaration: `procedure Read(var Buf;Count: LongInt); Virtual`

Visibility: default

Description: Read is an abstract method that should be overridden by descendent objects.

Read reads Count bytes from the stream into Buf. It updates the position pointer, increasing it's value with Count. Buf must be large enough to contain Count bytes.

Errors: No checking is done to see if Buf is large enough to contain Count bytes.

See also: TStream.Write (719), TStream.ReadStr (715), TStream.StrRead (713)

Listing: ./objectex/ex18.pp

```

program ex18;

{ Program to demonstrate the TStream.Read method }

Uses Objects;

Var Buf1, Buf2 : Array[1..1000] of Byte;
    I : longint;
    S : PMemoryStream;

begin
    For I:=1 to 1000 do
        Buf1[I]:=Random(1000);
    Buf2:=Buf1;
    S:=New(PMemoryStream, Init(100,10));
    S^.Write(Buf1, SizeOf(Buf1));
    S^.Seek(0);
    For I:=1 to 1000 do
        Buf1[I]:=0;
    S^.Read(Buf1, SizeOf(Buf1));
    For I:=1 to 1000 do
        If Buf1[I]<>buf2[i] then
            WriteLn('Buffer differs at position ',I);
    Dispose(S, Done);
end.

```

TStream.Write

Synopsis: Write a number of bytes to the stream.

Declaration: `procedure Write(var Buf;Count: LongInt); Virtual`

Visibility: default

Description: Write is an abstract method that should be overridden by descendent objects.

Write writes Count bytes to the stream from Buf. It updates the position pointer, increasing it's value with Count.

For an example, see TStream.Read (719).

Errors: No checking is done to see if Buf actually contains Count bytes.

See also: TStream.Read (719), TStream.WriteString (718), TStream.StrWrite (718)

TStream.CopyFrom

Synopsis: Copy data from another stream.

Declaration: `procedure CopyFrom(var S: TStream; Count: LongInt)`

Visibility: default

Description: `CopyFrom` reads `Count` bytes from stream `S` and stores them in the current stream. It uses the `Read` (719) method to read the data, and the `Write` (719) method to write in the current stream.

Errors: None.

See also: `TStream.Read` (719), `TStream.Write` (719)

Listing: `./objectex/ex19.pp`

Program `ex19`;

{ Program to demonstrate the TStream.CopyFrom function }

Uses `objects`;

Var `P` : `PString`;
 `L` : **String**;
 `S1,S2` : `PStream`;

begin
 `L:= 'Constant string line';`
 Writeln ('Writing to stream 1 : " ',`L`, ' " ');
 `S1:=New(PMemoryStream, Init(100,10));`
 `S2:=New(PMemoryStream, Init(100,10));`
 `S1^.WriteStr(@L);`
 `S1^.Seek(0);`
 Writeln ('Copying contents of stream 1 to stream 2 ');
 `S2^.Copyfrom(S1^,S1^.GetSize);`
 `S2^.Seek(0);`
 `P:=S2^.ReadStr;`
 `L:=P^;`
 DisposeStr(`P`);
 Dispose (`S1`, `Done`);
 Dispose (`S2`, `Done`);
 Writeln ('Read from stream 2 : " ',`L`, ' " ');
end.

20.16 TStringCollection**Description**

The `TStringCollection` object manages a sorted collection of pascal strings. To this end, it overrides the `Compare` (706) method of `TSortedCollection`, and it introduces methods to read/write strings from a stream.

Method overview

Page	Method	Description
721	Compare	Compare two strings in the collection.
722	FreeItem	Dispose a string in the collection from memory.
721	GetItem	Get string from the stream.
722	PutItem	Write a string to the stream.

TStringCollection.GetItem

Synopsis: Get string from the stream.

Declaration: `function GetItem(var S: TStream) : Pointer; Virtual`

Visibility: default

Description: `GetItem` reads a string from the stream `S` and returns a pointer to it. It doesn't insert the string in the collection.

This method is primarily introduced to be able to load and store the collection from and to a stream.

Errors: The errors returned are those of `TStream.ReadStr` ([715](#)).

See also: `TStringCollection.PutItem` ([722](#))

TStringCollection.Compare

Synopsis: Compare two strings in the collection.

Declaration: `function Compare(Key1: Pointer;Key2: Pointer) : Sw_Integer; Virtual`

Visibility: default

Description: `TStringCollection` overrides the `Compare` function so it compares the two keys as if they were pointers to strings. The compare is done case sensitive. It returns the following results:

-1 if the first string is alphabetically earlier than the second string.

0 if the two strings are equal.

1 if the first string is alphabetically later than the second string.

Errors: None.

See also: `TSortedCollection.Compare` ([706](#))

Listing: `./objectex/ex37.pp`

Program `ex37`;

{ Program to demonstrate the TStringCollection.Compare method }

Uses `Objects`;

Var `C : PStringCollection;`
`S : String;`
`I : longint;`

begin
`Randomize;`

```

C:=New(PStringCollection, Init(120,10));
C^.Duplicates:=True; { Duplicates allowed }
WriteLn ('Inserting 100 records at random places. ');
For I:=1 to 100 do
  begin
    Str(Random(100),S);
    S:='String with value '+S;
    C^.Insert(NewStr(S));
  end;
For I:=0 to 98 do
  With C^ do
    If Compare (At(i),At(I+1))=0 then
      WriteLn ('Duplicate string found at position ',i);
Dispose(C,Done);
end.

```

TStringCollection.FreeItem

Synopsis: Dispose a string in the collection from memory.

Declaration: `procedure FreeItem(Item: Pointer); Virtual`

Visibility: default

Description: `TStringCollection` overrides `FreeItem` so that the string pointed to by `Item` is disposed from memory.

Errors: None.

See also: `TCollection.FreeItem` ([681](#))

TStringCollection.PutItem

Synopsis: Write a string to the stream.

Declaration: `procedure PutItem(var S: TStream; Item: Pointer); Virtual`

Visibility: default

Description: `PutItem` writes the string pointed to by `Item` to the stream `S`.

This method is primarily used in the `Load` and `Store` methods, and should not be used directly.

Errors: Errors are those of `TStream.WriteString` ([718](#)).

See also: `TStringCollection.GetItem` ([721](#))

20.17 TStringList

Description

A `TStringList` object can be used to read a collection of strings stored in a stream. If you register this object with the `RegisterType` ([664](#)) function, you cannot register the `TStrListMaker` object.

Method overview

Page	Method	Description
723	Done	Clean up the instance
723	Get	Return a string by key name
723	Load	Load stringlist from stream.

TStringList.Load

Synopsis: Load stringlist from stream.

Declaration: constructor Load(var S: TStream)

Visibility: default

Description: The Load constructor reads the TStringList object from the stream S. It also reads the descriptions of the strings from the stream. The string descriptions are stored as an array of TStrIndexrec records, where each record describes a string on the stream. These records are kept in memory.

Errors: If an error occurs, a stream error is triggered.

See also: TStringList.Done ([723](#))

TStringList.Done

Synopsis: Clean up the instance

Declaration: destructor Done; Virtual

Visibility: default

Description: The Done destructor frees the memory occupied by the string descriptions, and destroys the object.

Errors: None.

See also: TStringList.Load ([723](#)), TObject.Done ([693](#))

TStringList.Get

Synopsis: Return a string by key name

Declaration: function Get(Key: Sw_Word) : String

Visibility: default

Description: Get reads the string with key Key from the list of strings on the stream, and returns this string. If there is no string with such a key, an empty string is returned.

Errors: If no string with key Key is found, an empty string is returned. A stream error may result if the stream doesn't contain the needed strings.

See also: TStrListMaker.Put ([724](#))

20.18 TStrListMaker

Description

The `TStrListMaker` object can be used to generate a stream with strings, which can be read with the `TStringList` object. If you register this object with the `RegisterType` (664) function, you cannot register the `TStringList` object.

Method overview

Page	Method	Description
724	<code>Done</code>	Clean up the instance and free all related memory.
724	<code>Init</code>	Instantiate a new instance of <code>TStrListMaker</code>
724	<code>Put</code>	Add a new string to the list with associated key.
725	<code>Store</code>	Write the strings to the stream.

TStrListMaker.Init

Synopsis: Instantiate a new instance of `TStrListMaker`

Declaration: constructor `Init(AStrSize: Sw_Word; AIndexSize: Sw_Word)`

Visibility: default

Description: The `Init` constructor creates a new instance of the `TStrListMaker` object. It allocates `AStrSize` bytes on the heap to hold all the strings you wish to store. It also allocates enough room for `AINdexSize` key description entries (of the type `TStrIndexrec`).

`AStrSize` must be large enough to contain all the strings you wish to store. If not enough memory is allocated, other memory will be overwritten. The same is true for `AINdexSize` : maximally `AINdexSize` strings can be written to the stream.

Errors: None.

See also: `TObject.Init` (692), `TStrListMaker.Done` (724)

TStrListMaker.Done

Synopsis: Clean up the instance and free all related memory.

Declaration: destructor `Done; Virtual`

Visibility: default

Description: The `Done` destructor de-allocates the memory for the index description records and the string data, and then destroys the object.

Errors: None.

See also: `TObject.Done` (693), `TStrListMaker.Init` (724)

TStrListMaker.Put

Synopsis: Add a new string to the list with associated key.

Declaration: procedure `Put(Key: Sw_Word; S: String)`

Visibility: default

Description: `Put` adds the string `S` with key `Key` to the collection of strings. This action doesn't write the string to a stream. To write the strings to the stream, see the `Store` (725) method.

Errors: None.

See also: `TStrListMaker.Store` (725)

TStrListMaker.Store

Synopsis: Write the strings to the stream.

Declaration: `procedure Store(var S: TStream)`

Visibility: default

Description: `Store` writes the collection of strings to the stream `S`. The collection can then be read with the `TStringList` object.

Errors: A stream error may occur when writing the strings to the stream.

See also: `TStringList.Load` (723), `TStrListMaker.Put` (724)

20.19 TUnSortedStrCollection

Description

The `TUnSortedStrCollection` object manages an unsorted list of strings. To this end, it overrides the `TStringCollection.Insert` (720) method to add strings at the end of the collection, rather than in the alphabetically correct position.

Take care, the `Search` (707) and `IndexOf` (673) methods will not work on an unsorted string collection.

Method overview

Page	Method	Description
725	<code>Insert</code>	Insert a new string in the collection.

TUnSortedStrCollection.Insert

Synopsis: Insert a new string in the collection.

Declaration: `procedure Insert(Item: Pointer); Virtual`

Visibility: default

Description: `Insert` inserts a string at the end of the collection, instead of on its alphabetical place, resulting in an unsorted collection of strings.

Errors: None.

See also: `TCollection.Insert` (679)

Listing: `./objectex/ex39.pp`

```
Program ex39;  
  
{ Program to demonstrate the TUnsortedStrCollection.Insert method }  
  
Uses Objects , Strings ;  
  
Var C : PUnsortedStrCollection ;  
      S : String ;  
      I : longint ;  
      P : Pchar ;  
  
begin  
  Randomize ;  
  C:=New(PUnsortedStrCollection , Init(120,10));  
  Writeln ( 'Inserting 100 records at random places.' );  
  For I:=1 to 100 do  
    begin  
      Str(Random(100),S);  
      S:='String with value '+S;  
      C^.Insert(NewStr(S));  
    end ;  
  For I:=0 to 99 do  
    Writeln ( I:2,': ',PString(C^.At(i))^ );  
  Dispose(C,Done);  
end.
```

Chapter 21

Reference for unit 'objpas'

21.1 Overview

The `objpas` unit is meant for compatibility with Object Pascal as implemented by Delphi. The unit is loaded automatically by the Free Pascal compiler whenever the `Delphi` or `objfpc` mode is entered, either through the command line switches `-Sd` or `-Sh` or with the `{ $MODE DELPHI }` or `{ $MODE OBJFPC }` directives.

It redefines some basic pascal types, introduces some functions for compatibility with Delphi's system unit, and introduces some methods for the management of the resource string tables.

21.2 Constants, types and variables

Constants

`MaxInt = MaxLongint`

Maximum value for Integer (727) type.

Types

`integer = LongInt`

In OBJPAS mode and in DELPHI mode, an Integer has a size of 32 bit. In TP or regular FPC mode, an integer is 16 bit.

`IntegerArray = Array[0..$effffff] of Integer`

Generic array of integer (727)

`PIntegerArray = ^IntegerArray`

Pointer to TIntegerArray (728) type.

`PointerArray = Array[0..512*1024*1024-2] of Pointer`

Generic Array of pointers.

`PPointerArray = ^PointerArray`

Pointer to PointerArray ([727](#))

`PResStringRec = ^AnsiString`

Pointer to ansistring (Delphi compatibility).

`PString = PAnsiString`

Pointer to ansistring type.

`TBoundArray = Array[] of Integer`

Array of integer, used in interfaces.

`TIntegerArray = IntegerArray`

Alias for IntegerArray ([727](#))

`TPointerArray = PointerArray`

Alias for PointerArray ([727](#))

`TResourceIterator = function(Name: AnsiString; Value: AnsiString;
Hash: LongInt) : AnsiString`

The resource string tables can be managed with a callback function which the user must provide:
`TResourceIterator`.

`TResStringRec = AnsiString`

Ansistring record in resource table (Delphi compatibility).

21.3 Procedures and functions

AssignFile

Synopsis: Assign text or untyped file

Declaration: `procedure AssignFile(var f: File; const Name: String)`
`procedure AssignFile(var f: File; p: pchar)`
`procedure AssignFile(var f: File; c: Char)`
`procedure AssignFile(var t: Text; const s: String)`
`procedure AssignFile(var t: Text; p: pchar)`
`procedure AssignFile(var t: Text; c: Char)`
`procedure AssignFile(var f: TypedFile; const Name: String)`
`procedure AssignFile(var f: TypedFile; p: pchar)`
`procedure AssignFile(var f: TypedFile; c: Char)`

Visibility: default

Description: AssignFile is completely equivalent to the system unit's Assign (977) function: It assigns Name to a function of any type (FileType can be Text or a typed or untyped File variable). Name can be a string, a single character or a PChar.

It is most likely introduced to avoid confusion between the regular Assign (977) function and the Assign method of TPersistent in the Delphi VCL.

Errors: None.

See also: CloseFile (729), #rtl.system.Assign (977), #rtl.system.Reset (1033), #rtl.system.Rewrite (1034), #rtl.system.Append (975)

Listing: ./refex/ex88.pp

Program Example88;

{ Program to demonstrate the AssignFile and CloseFile functions. }

{ \$MODE Delphi }

Var F : text;

begin

 AssignFile(F, 'textfile.tmp');

 Rewrite(F);

 Writeln (F, 'This is a silly example of AssignFile and CloseFile.');

 CloseFile(F);

end.

CloseFile

Synopsis: Close text or untyped file

Declaration: procedure CloseFile(var f: File)
 procedure CloseFile(var t: Text)

Visibility: default

Description: CloseFile flushes and closes a file F of any file type. F can be Text or a typed or untyped File variable. After a call to CloseFile, any attempt to write to the file F will result in an error.

It is most likely introduced to avoid confusion between the regular Close (982) function and the Close method of TForm in the Delphi VCL.

for an example, see AssignFile (728).

Errors: None.

See also: #rtl.system.Close (982), AssignFile (728), #rtl.system.Reset (1033), #rtl.system.Rewrite (1034), #rtl.system.Append (975)

GetResourceStringCurrentValue

Synopsis: Return current value of resourcestring

Declaration: function GetResourceStringCurrentValue(TableIndex: LongInt;
 StringIndex: LongInt) : AnsiString

Visibility: default

Description: `GetResourceStringCurrentValue` returns the current value of the resourcestring in table `TableIndex` with index `StringIndex`.

The current value depends on the system of internationalization that was used, and which language is selected when the program is executed.

Errors: If either `TableIndex` or `StringIndex` are out of range, then a empty string is returned.

See also: `SetResourceStrings` (734), `GetResourceStringDefaultValue` (730), `GetResourceStringHash` (731), `GetResourceStringName` (731), `ResourceStringTableCount` (734), `ResourceStringCount` (734)

Listing: `./refex/ex90.pp`

Program `Example90`;

```
{ Program to demonstrate the GetResourceStringCurrentValue function. }
{$Mode Delphi}
```

```
ResourceString
```

```
    First = 'First string';
    Second = 'Second String';
```

```
Var I,J : Longint;
```

```
begin
    { Print current values of all resourcestrings }
    For I:=0 to ResourceStringTableCount-1 do
        For J:=0 to ResourceStringCount(i)-1 do
            WriteLn (I, ', ', J, ' : ', GetResourceStringCurrentValue(I,J));
end.
```

GetResourceStringDefaultValue

Synopsis: Return default (original) value of resourcestring

Declaration: `function GetResourceStringDefaultValue(TableIndex: LongInt;
StringIndex: LongInt) : AnsiString`

Visibility: default

Description: `GetResourceStringDefaultValue` returns the default value of the resourcestring in table `TableIndex` with index `StringIndex`.

The default value is the value of the string that appears in the source code of the programmer, and is compiled into the program.

Errors: If either `TableIndex` or `StringIndex` are out of range, then a empty string is returned.

See also: `SetResourceStrings` (734), `GetResourceStringCurrentValue` (729), `GetResourceStringHash` (731), `GetResourceStringName` (731), `ResourceStringTableCount` (734), `ResourceStringCount` (734)

Listing: `./refex/ex91.pp`

Program `Example91`;

```
{ Program to demonstrate the GetResourceStringDefaultValue function. }
{$Mode Delphi}
```

ResourceString

```
First = 'First string';
Second = 'Second String';
```

```
Var I,J : Longint;
```

```
begin
  { Print default values of all resourcestrings }
  For I:=0 to ResourceStringTableCount-1 do
    For J:=0 to ResourceStringCount(I)-1 do
      WriteLn (I, ',', J, ' : ', GetResourceStringDefaultValue(I,J));
end.
```

GetResourceStringHash

Synopsis: Return hash value of resource string

Declaration: function GetResourceStringHash(TableIndex: LongInt;StringIndex: LongInt)
: LongInt

Visibility: default

Description: GetResourceStringHash returns the hash value associated with the resource string in table TableIndex, with index StringIndex.

The hash value is calculated from the default value of the resource string in a manner that gives the same result as the GNU `gettext` mechanism. It is stored in the resourcestring tables, so retrieval is faster than actually calculating the hash for each string.

For an example, see Hash (732).

Errors: If either TableIndex or StringIndex is zero, 0 is returned.

See also: Hash (732), SetResourceStrings (734), GetResourceStringDefaultValue (730), GetResourceStringHash (731), GetResourceStringName (731), ResourceStringTableCount (734), ResourceStringCount (734)

GetResourceStringName

Synopsis: Return name of resource string.

Declaration: function GetResourceStringName(TableIndex: LongInt;StringIndex: LongInt)
: Ansistring

Visibility: default

Description: GetResourceStringName returns the name of the resourcestring in table TableIndex with index StringIndex. The name of the string is always the unit name in which the string was declared, followed by a period and the name of the constant, all in lowercase.

If a unit `MyUnit` declares a resourcestring `MyTitle` then the name returned will be `myunit.mytitle`. A resourcestring in the program file will have the name of the program prepended.

The name returned by this function is also the name that is stored in the resourcestring file generated by the compiler.

Strictly speaking, this information isn't necessary for the functioning of the program, it is provided only as a means to easier translation of strings.

Errors: If either `TableIndex` or `StringIndex` is zero, an empty string is returned.

See also: [SetResourceStrings \(734\)](#), [GetResourceStringDefaultValue \(730\)](#), [GetResourceStringHash \(731\)](#), [GetResourceStringName \(731\)](#), [ResourceStringTableCount \(734\)](#), [ResourceStringCount \(734\)](#)

Listing: ./refex/ex92.pp

Program Example92;

```
{ Program to demonstrate the GetResourceStringName function. }
{$Mode Delphi}
```

ResourceString

```
First = 'First string';
Second = 'Second String';
```

Var I,J : Longint;

```
begin
  { Print names of all resourcestrings }
  For I:=0 to ResourceStringTableCount-1 do
    For J:=0 to ResourceStringCount(i)-1 do
      WriteLn (I, ', ', J, ' : ', GetResourceStringName(I,J));
end.
```

Hash

Synopsis: Create GNU Gettext hash value for a string

Declaration: `function Hash(S: AnsiString) : LongInt`

Visibility: default

Description: Hash calculates the hash value of the string S in a manner that is compatible with the GNU gettext hash value for the string. It is the same value that is stored in the Resource string tables, and which can be retrieved with the [GetResourceStringHash \(731\)](#) function call.

Errors: None. In case the calculated hash value should be 0, the returned result will be -1.

See also: [GetResourceStringHash \(731\)](#)

Listing: ./refex/ex93.pp

Program Example93;

```
{ Program to demonstrate the Hash function. }
{$Mode Delphi}
```

ResourceString

```
First = 'First string';
Second = 'Second String';
```

Var I,J : Longint;

```
begin
  For I:=0 to ResourceStringTableCount-1 do
```

```

For J:=0 to ResourceStringCount(i)-1 do
  If Hash( GetResourceStringDefaultValue(I,J))
    <>GetResourceStringHash(I,J) then
    WriteLn ( 'Hash mismatch at ',I,', ',J)
  else
    WriteLn ( 'Hash ( ',I,', ',J, ' ) matches. ');
end.

```

LoadResString

Synopsis: Load resource string

Declaration: function LoadResString(p: PResStringRec) : AnsiString

Visibility: default

ParamStr

Synopsis: Return command-line parameter

Declaration: function ParamStr(Param: Integer) : Ansistring

Visibility: default

Description: ParamStr returns the Param-th command-line parameter as an AnsiString. The system unit Paramstr ([733](#)) function limits the result to 255 characters, and is overridden with this function.

The zeroeth command-line parameter contains the path of the executable. On some operating systems (BSD) it may be simply the command as typed on the command-line, because the OS does not offer a method to retrieve the full binary name.

For an example, see #rtl.system.Paramstr ([1025](#)).

Errors: In case Param is an invalid value, an empty string is returned.

See also: Paramstr ([733](#))

ResetResourceTables

Synopsis: Restore all resource strings to their declared values

Declaration: procedure ResetResourceTables

Visibility: default

Description: ResetResourceTables resets all resource strings to their default (i.e. as in the source code) values.

Normally, this should never be called from a user's program. It is called in the initialization code of the objpas unit. However, if the resourcetables get messed up for some reason, this procedure will fix them again.

Errors: None.

See also: SetResourceStrings ([734](#)), GetResourceStringDefaultValue ([730](#)), GetResourceStringHash ([731](#)), GetResourceStringName ([731](#)), ResourceStringTableCount ([734](#)), ResourceStringCount ([734](#))

ResourceStringCount

Synopsis: Return number of resource strings in table

Declaration: `function ResourceStringCount (TableIndex: LongInt) : LongInt`

Visibility: default

Description: `ResourceStringCount` returns the number of resource strings in the table with index `TableIndex`. The strings in a particular table are numbered from 0 to `ResourceStringCount-1`, i.e. they're zero based.

For an example, see `GetResourceStringDefaultValue` (730)

Errors: If an invalid `TableIndex` is given, -1 is returned.

See also: `SetResourceStrings` (734), `GetResourceStringCurrentValue` (729), `GetResourceStringDefaultValue` (730), `GetResourceStringHash` (731), `GetResourceStringName` (731), `ResourceStringTableCount` (734)

ResourceStringTableCount

Synopsis: Return number of resource string tables

Declaration: `function ResourceStringTableCount : LongInt`

Visibility: default

Description: `ResourceStringTableCount` returns the number of resource string tables; this may be zero if no resource strings are used in a program.

The tables are numbered from 0 to `ResourceStringTableCount-1`, i.e. they're zero based.

For an example, see `GetResourceStringDefaultValue` (730)

Errors:

See also: `SetResourceStrings` (734), `GetResourceStringDefaultValue` (730), `GetResourceStringHash` (731), `GetResourceStringName` (731), `ResourceStringCount` (734)

SetResourceStrings

Synopsis: Set values of all resource strings.

Declaration: `procedure SetResourceStrings (SetFunction: TResourceIterator)`

Visibility: default

Description: `SetResourceStrings` calls `SetFunction` for all resource strings in the resource string tables and sets the resource string's current value to the value returned by `SetFunction`.

The `Name`, `Value` and `Hash` parameters passed to the iterator function are the values stored in the tables.

Errors: None.

See also: `GetResourceStringCurrentValue` (729), `GetResourceStringDefaultValue` (730), `GetResourceStringHash` (731), `GetResourceStringName` (731), `ResourceStringTableCount` (734), `ResourceStringCount` (734)

Listing: `./refex/ex95.pp`

Program Example95;

```
{ Program to demonstrate the SetResourceStrings function. }
{$Mode objfpc}
```

ResourceString

```
First = 'First string';
Second = 'Second String';
```

```
Var I,J : Longint;  
    S : AnsiString;
```

Function Translate (Name, Value : AnsiString; Hash : longint): AnsiString;

```
begin
  WriteIn ( 'Translate ( ',Name,' ) => ',Value);
  Write   ( '→' );
  ReadIn  ( Result );
end;
```

```
begin
  SetResourceStrings ( @Translate );
  WriteLn ( 'Translated strings : ' );
  For I:=0 to ResourceStringTableCount-1 do
    For J:=0 to ResourceStringCount(i)-1 do
      begin
        WriteLn ( GetResourceStringDefaultValue ( I , J ) );
        WriteLn ( 'Translates to : ' );
        WriteLn ( GetResourceStringCurrentValue ( I , J ) );
      end;
```

```
end.
```

SetResourceStringValue

Synopsis: Set value of a resource string

```
Declaration: function SetResourceStringValue(TableIndex: LongInt;
                                             StringIndex: LongInt;Value: Ansistring;
                                             : Boolean
```

Visibility: default

Description: SetResourceStringValue assigns Value to the resource string in table TableIndex with index StringIndex.

Errors:

See also: [SetResourceStrings \(734\)](#), [GetResourceStringCurrentValue \(729\)](#), [GetResourceStringDefaultValue \(730\)](#), [GetResourceStringHash \(731\)](#), [GetResourceStringName \(731\)](#), [ResourceStringTableCount \(734\)](#), [ResourceStringCount \(734\)](#)

Listing: ./refex/ex94.pp

Program Example94;

```
{ Program to demonstrate the SetResourceStringValue function. }
```

```
{ $Mode Delphi }
```

```
ResourceString
```

```
    First  = 'First string';  
    Second = 'Second String';
```

```
Var I,J : Longint;  
    S : AnsiString;
```

```
begin
```

```
    { Print current values of all resourcestrings }
```

```
    For I:=0 to ResourceStringTableCount-1 do
```

```
        For J:=0 to ResourceStringCount(i)-1 do
```

```
            begin
```

```
                WriteLn ( 'Translate => ', GetResourceStringDefaultValue(I,J));
```

```
                Write    ('->');
```

```
                ReadLn(S);
```

```
                SetResourceStringValue(I,J,S);
```

```
            end;
```

```
    WriteLn ( 'Translated strings : ');
```

```
    For I:=0 to ResourceStringTableCount-1 do
```

```
        For J:=0 to ResourceStringCount(i)-1 do
```

```
            begin
```

```
                WriteLn ( GetResourceStringDefaultValue(I,J));
```

```
                WriteLn ( 'Translates to : ');
```

```
                WriteLn ( GetResourceStringCurrentValue(I,J));
```

```
            end;
```

```
end.
```

Chapter 22

Reference for unit 'oldlinux'

22.1 Utility routines

Auxiliary functions that are useful in connection with the other functions.

Table 22.1:

Name	Description
CreateShellArgV (797)	Create an array of pchars from string
EpochToLocal (800)	Convert epoch time to local time
FD_Clr (810)	Clear item of select filedescriptors
FD_IsSet (811)	Check item of select filedescriptors
FD_Set (811)	Set item of select filedescriptors
FD_ZERO (811)	Clear all items in select filedecriptors
LocalToEpoch (830)	Convert local time to epoch time
MMap (832)	Map a file into memory
MUnMap (834)	Unmap previously mapped memory file
Octal (836)	Convert octal to digital
S_ISBLK (852)	Check file mode for block device
S_ISCHR (852)	Check file mode for character device
S_ISDIR (852)	Check file mode for directory
S_ISFIFO (853)	Check file mode for FIFO
S_ISLNK (853)	Check file mode for symboloc link
S_ISREG (854)	Check file mode for regular file
S_ISSOCK (854)	Check file mode for socket
StringToPPchar (848)	Create an array of pchars from string

22.2 Terminal functions

Functions for controlling the terminal to which the process is connected.

22.3 System information

Functions for retrieving system information such as date and time.

Table 22.2:

Name	Description
CFMakeRaw (792)	Set terminal to raw mode
CFSetISpeed (792)	Set terminal reading speed
CFSetOSpeed (792)	Set terminal writing speed
IOCTL (827)	General IO control call
IsATTY (828)	See if filedescriptor is a terminal
TCDrain (854)	Wait till all output was written
TCFlow (854)	Suspend transmission or receipt of data
TCFlush (855)	Discard data written to terminal
TCGetAttr (855)	Get terminal attributes
TCGetPGrp (856)	Return PID of foreground process
TCSendBreak (856)	Send data for specific time
TCSetAttr (857)	Set terminal attributes
TCSetPGrp (857)	Set foreground process
TTYName (858)	Name of tty file

Table 22.3:

Name	Description
GetDate (817)	Return system date
GetDateTime (818)	Return system date and time
GetDomainName (818)	Return system domain name
GetEpochTime (820)	Return epoch time
GetHostName (822)	Return system host name
GetLocalTimezone (822)	Return system timezone
GetTime (824)	Return system time
GetTimeOfDay (825)	Return system time
GetTimezoneFile (825)	Return name of timezone file
ReadTimezoneFile (840)	Read timezone file contents
SysInfo (851)	Return general system information
Uname (858)	Return system information

22.4 Signals

Functions for managing and responding to signals.

22.5 Process handling

Functions for managing processes and programs.

22.6 Directory handling routines

Functions for reading and searching directories.

Table 22.4:

Name	Description
Alarm (788)	Send alarm signal to self
Kill (828)	Send arbitrary signal to process
pause (837)	Wait for signal to arrive
SigAction (844)	Set signal action
Signal (846)	Set signal action
SigPending (847)	See if signals are waiting
SigProcMask (847)	Set signal processing mask
SigRaise (847)	Send signal to self
SigSuspend (848)	Sets signal mask and waits for signal
NanoSleep (834)	Waits for a specific amount of time

22.7 Pipes, FIFOs and streams

Functions for creating and managing pipes.

22.8 General File handling routines

Functions for handling files on disk.

22.9 File Input/Output routines

Functions for handling file input/output.

22.10 Overview

This document describes the LINUX unit for Free Pascal. The unit was written by Michael van Canneyt. It works only on the Linux/BSD operating systems.

22.11 Constants, types and variables

Constants

B0 = \$00000000

B110 = \$00000003

B115200 = \$0001002

B1200 = \$00000009

Table 22.5:

Name	Description
Clone (795)	Create a thread
Execl (800)	Execute process with command-line list
Execle (801)	Execute process with command-line list and environment
Execlp (802)	Search in path and execute process with command list
Execv (803)	Execute process
Execve (804)	Execute process with environment
Execvp (805)	Search in path and execute process
Fork (813)	Spawn child process
GetEGid (819)	Get effective group id
GetEnv (819)	Get environment variable
GetEUid (820)	Get effective user id
GetGid (821)	Get group id
GetPid (823)	Get process id
GetPPid (823)	Get parent process id
GetPriority (824)	Get process priority
GetUid (825)	Get user id
Nice (835)	Change priority of process
SetPriority (843)	Change priority of process
Shell (844)	Execute shell command
WaitPid (860)	Wait for child process to terminate

Table 22.6:

Name	Description
CloseDir (797)	Close directory handle
Glob (826)	Return files matching a search expression
GlobFree (827)	Free result of Glob
OpenDir (836)	Open directory for reading
ReadDir (839)	Read directory entry
SeekDir (841)	Seek directory
TellDir (857)	Seek directory

B134 = \$0000004

B150 = \$0000005

B1800 = \$000000A

B19200 = \$000000E

B200 = \$0000006

B230400 = \$0001003

Table 22.7:

Name	Description
AssignPipe (789)	Create a pipe
AssignStream (790)	Create pipes to program's input and output
MkFifo (832)	Make a fifo
PClose (838)	Close a pipe
POpen (838)	Open a pipe for to program's input or output

Table 22.8:

Name	Description
Access (787)	Check access rights on file
BaseName (791)	Return name part of file
Chown (794)	Change owner of file
Chmod (792)	Change access rights on file
DirName (798)	Return directory part of file
FSplit (815)	Split filename in parts
FExpand (811)	Return full-grown filename
FLock (812)	Set lock on a file
FNMatch (812)	Match filename to searchpattern
FSearch (814)	Search for a file in a path
FStat (815)	Return filesystem information
FStat (816)	Return file information
FRename (813)	Rename file
LStat (831)	Return information on a link
Link (829)	Create a link
ReadLink (839)	Read contents of a symbolic link
SymLink (849)	Create a symbolic link
Umask (858)	Set the file creation mask
UnLink (859)	Remove a file
Utime (859)	Change file timestamps

B2400 = \$000000B

B300 = \$0000007

B38400 = \$000000F

B460800 = \$0001004

B4800 = \$000000C

B50 = \$0000001

B57600 = \$0001001

Table 22.9:

Name	Description
Dup (798)	Duplicate a file handle
Dup2 (799)	Copy one file handle to another
Fcntl (806)	General file control
fdClose (807)	Close file descriptor
fdFlush (807)	Flush file descriptor
fdOpen (807)	Open new file descriptor
fdRead (808)	Read from file descriptor
fdSeek (809)	Position in file
fdTruncate (810)	Truncate file
fdWrite (810)	Write to file descriptor
GetFS (821)	Get file descriptor of pascal file
Select (841)	Wait for input from file descriptor
SelectText (842)	Wait for input from pascal file

B600 = \$0000008

B75 = \$0000002

B9600 = \$000000D

BRKINT = \$0000002

BS0 = \$0000000

BS1 = \$0002000

BSDLY = \$0002000

CBAUD = \$000100F

CBAUDEX = \$0001000

CIBAUD = \$100F0000

CLOCAL = \$0000800

CLONE_FILES = \$00000400

Clone (795) option: open files shared between processes

CLONE_FS = \$00000200

Clone (795) option: fs info shared between processes

CLONE_PID = \$00001000

Clone (795) option: PID shared between processes

CLONE_SIGHAND = \$00000800

Clone (795) option: signal handlers shared between processes

CLONE_VM = \$00000100

Clone (795) option: VM shared between processes

CMSPAR = \$40000000

CR0 = \$00000000

CR1 = \$0000200

CR2 = \$0000400

CR3 = \$0000600

CRDLY = \$0000600

CREAD = \$0000080

CRTSCTS = \$80000000

CS5 = \$0000000

CS6 = \$0000010

CS7 = \$0000020

CS8 = \$0000030

CSIGNAL = \$000000ff

Clone (795) option: Signal mask to be sent at exit

CSize = \$0000030

CSTOPB = \$0000040

ECHO = \$0000008

ECHOCTL = \$0000200

ECHOE = \$0000010

ECHOK = \$0000020

ECHOKE = \$0000800

ECHONL = \$0000040

ECHOPRT = \$0000400

EXTA = B19200

EXTB = B38400

FF0 = \$0000000

FF1 = \$0008000

FFDLY = \$0008000

FIOASYNC = \$5452

FIOCLEX = \$5451

FIONBIO = \$5421

FIONCLEX = \$5450

FIONREAD = \$541B

FLUSHO = \$0001000

fs_ext = \$137d

File system type (FSStat (815)): (ext) Extended

fs_ext2 = \$ef53

File system type (FSStat (815)): (ext2) Second extended

fs_iso = \$9660

File system type (FSStat (815)): ISO 9660

fs_minix = \$137f

File system type (FSStat (815)): Minix

fs_minix_30 = \$138f

File system type (FSStat (815)): Minix 3.0

fs_minix_V2 = \$2468

File system type (FSStat (815)): Minix V2

fs_msdos = \$4d44

File system type (FSStat (815)): MSDOS (FAT)

fs_nfs = \$6969

File system type (FSStat (815)): NFS

fs_old_ext2 = \$ef51

File system type (FSStat (815)): (ext2) Old second extended

fs_proc = \$9fa0

File system type (FSStat (815)): PROC fs

fs_xia = \$012FD16D

File system type (FSStat (815)): XIA

F_GetFd = 1

FCntl (806) command: Get close-on-exec flag

F_GetFl = 3

FCntl (806) command: Get filedescriptor flags

F_GetLk = 5

FCntl (806) command: Get lock

F_GetOwn = 9

FCntl (806) command: get owner of filedescriptor events

F_OK = 0

Access (787) call test: file exists.

F_SetFd = 2

FCntl (806) command: Set close-on-exec flag

F_SetFl = 4

FCntl (806) command: Set filedescriptor flags

F_SetLk = 6

FCntl (806) command: Set lock

F_SetLkW = 7

FCntl (806) command: Test lock

F_SetOwn = 8

FCntl (806) command: Set owner of filedescriptor events

HUPCL = \$0000400

ICANON = \$0000002

ICRNL = \$0000100

IEXTEN = \$0008000

IGNBRK = \$0000001

IGNCR = \$0000080

IGNPAR = \$0000004

IMAXBEL = \$0002000

INLCR = \$0000040

INPCK = \$0000010

IOctl_TCGETS = \$5401

IOCTL call number: get Terminal Control settings

ISIG = \$0000001

ISTRIP = \$0000020

IUCLC = \$0000200

IXANY = \$0000800

IXOFF = \$0001000

IXON = \$0000400

LOCK_EX = 2

Flock (812) Exclusive lock

LOCK_NB = 4

Flock (812) Non-blocking operation

LOCK_SH = 1

Flock (812) Shared lock

LOCK_UN = 8

Flock (812) unlock

MAP_ANONYMOUS = \$20

MMap (832) map type: Don't use a file

MAP_DENYWRITE = \$800

MMap (832) option: Ignored.

MAP_EXECUTABLE = \$1000

MMap (832) option: Ignored.

MAP_FIXED = \$10

MMap (832) map type: Interpret addr exactly

MAP_GROWSDOWN = \$100

MMap (832) option: Memory grows downward (like a stack)

MAP_LOCKED = \$2000

MMap (832) option: lock the pages in memory.

MAP_NORESERVE = \$4000

MMap (832) option: Do not reserve swap pages for this memory.

MAP_PRIVATE = 2

MMap (832) map type: Changes are private

MAP_SHARED = \$1

MMap (832) map type: Share changes

MAP_TYPE = \$f

MMap (832) map type: Bitmask for type of mapping

MINSIGSTKSZ = 2048

NCC = 8

Number of control characters in termio (783) record.

NCCS = 32

Number of control characters in termios (784) record.

NL0 = \$00000000

NL1 = \$00000100

NLDLY = \$00000100

NOFLSH = \$00000080

OCRNL = \$00000008

OFDEL = \$00000080

OFILL = \$0000040

OLCUC = \$0000002

ONLCR = \$0000004

ONLRET = \$0000020

ONOCR = \$0000010

Open_Accmode = 3

Bitmask to determine access mode in open flags.

Open_Append = 2 shl 9

File open mode: Append to file

Open_Creat = 1 shl 6

File open mode: Create if file does not yet exist.

Open_Direct = 4 shl 12

File open mode: Minimize caching effects

Open_Directory = 2 shl 15

File open mode: File must be directory.

Open_Excl = 2 shl 6

File open mode: Open exclusively

Open_LargeFile = 1 shl 15

File open mode: Open for 64-bit I/O

Open_NDelay = Open_NonBlock

File open mode: Alias for Open_NonBlock (750)

Open_NoCtty = 4 shl 6

File open mode: No TTY control.

Open_NoFollow = 4 shl 15

File open mode: Fail if file is symbolic link.

Open_NonBlock = 4 shl 9

File open mode: Open in non-blocking mode

Open_RdOnly = 0

File open mode: Read only

Open_RdWr = 2

File open mode: Read/Write

Open_Sync = 1 shl 12

File open mode: Write to disc at once

Open_Trunc = 1 shl 9

File open mode: Truncate file to length 0

Open_WrOnly = 1

File open mode: Write only

OPOST = \$0000001

PARENB = \$0000100

PARMRK = \$0000008

PARODD = \$0000200

PENDIN = \$0004000

Prio_PGrp = 1

Get/set process group priority

Prio_Process = 0

Get/Set process priority

Prio_User = 2

Get/set user priority

PROT_EXEC = \$4

MMap ([832](#)) memory access: page can be executed

PROT_NONE = 0

MMap (832) memory access: page can not be accessed

PROT_READ = 1

MMap (832) memory access: page can be read

PROT_WRITE = 2

MMap (832) memory access: page can be written

P_IN = 1

Input file descriptor of pipe pair.

P_OUT = 2

Output file descriptor of pipe pair.

R_OK = 4

Access (787) call test: read allowed

SA_INTERRUPT = 0x20000000

Sigaction options: ?

SA_NOCLDSTOP = 1

Sigaction options: Do not receive notification when child processes stop

SA_NOMASK = 0x40000000

Sigaction options: Do not prevent the signal from being received when it is handled.

SA_ONESHOT = 0x80000000

Sigaction options: Restore the signal action to the default state.

SA_ONSTACK = SA_STACK

Socket option

SA_RESTART = 0x10000000

Sigaction options: Provide behaviour compatible with BSD signal semantics

SA_SHIRQ = 0x04000000

Sigaction options: ?

SA_STACK = 0x08000000

Sigaction options: Call the signal handler on an alternate signal stack.

Seek_Cur = 1

Seek option: Set position relative to current position.

Seek_End = 2

Seek option: Set position relative to end of file.

Seek_set = 0

Seek option: Set absolute position.

SIGABRT = 6

Signal: ABRT (Abort)

SIGALRM = 14

Signal: ALRM (Alarm clock)

SIGBUS = 7

Signal: BUS (bus error)

SIGCHLD = 17

Signal: CHLD (child status changed)

SIGCONT = 18

Signal: CONT (Continue)

SIGFPE = 8

Signal: FPE (Floating point error)

SIGHUP = 1

Signal: HUP (Hangup)

SIGILL = 4

Signal: ILL (Illegal instruction)

SIGINT = 2

Signal: INT (Interrupt)

SIGIO = 29

Signal: IO (I/O operation possible)

SIGIOT = 6

Signal: IOT (IOT trap)

SIGKILL = 9

Signal: KILL (unblockable)

SIGPIPE = 13

Signal: PIPE (Broken pipe)

SIGPOLL = SIGIO

Signal: POLL (Pollable event)

SIGPROF = 27

Signal: PROF (Profiling alarm)

SIGPWR = 30

Signal: PWR (power failure restart)

SIGQUIT = 3

Signal: QUIT

SIGSEGV = 11

Signal: SEGV (Segmentation violation)

SIGSTKFLT = 16

Signal: STKFLT (Stack Fault)

SIGSTKSZ = 8192

Signal Stack size error

SIGSTOP = 19

Signal: STOP (Stop, unblockable)

SIGTerm = 15

Signal: TERM (Terminate)

SIGTRAP = 5

Signal: TRAP (Trace trap)

SIGTSTP = 20

Signal: TSTP (keyboard stop)

SIGTTIN = 21

Signal: TTIN (Terminal input, background)

SIGTTOU = 22

Signal: TTOU (Terminal output, background)

SIGUNUSED = 31

Signal: Unused

SIGURG = 23

Signal: URG (Socket urgent condition)

SIGUSR1 = 10

Signal: USR1 (User-defined signal 1)

SIGUSR2 = 12

Signal: USR2 (User-defined signal 2)

SIGVTALRM = 26

Signal: VTALRM (Virtual alarm clock)

SIGWINCH = 28

Signal: WINCH (Window/Terminal size change)

SIGXCPU = 24

Signal: XCPU (CPU limit exceeded)

SIGXFSZ = 25

Signal: XFSZ (File size limit exceeded)

SIG_BLOCK = 0

Sigprocmask flags: Add signals to the set of blocked signals.

SIG_DFL = 0

Signal handler: Default signal handler

SIG_ERR = -1

Signal handler: error

SIG_IGN = 1

Signal handler: Ignore signal

SIG_SETMASK = 2

Sigprocmask flags: Set of blocked signals is given.

`SIG_UNBLOCK = 1`

Sigprocmask flags: Remove signals from the set set of blocked signals.

`SI_PAD_SIZE = ((128 / sizeof (longint)) - 3)`

Signal information record pad bytes size. Do not use.

`SS_DISABLE = 2`

Socket options

`SS_ONSTACK = 1`

Socket options

`STAT_IFBLK = $6000`

File (stat (782) record) mode: Block device

`STAT_IFCHR = $2000`

File (stat (782) record) mode: Character device

`STAT_IFDIR = $4000`

File (stat (782) record) mode: Directory

`STAT_IFIFO = $1000`

File (stat (782) record) mode: FIFO

`STAT_IFLNK = $a000`

File (stat (782) record) mode: Link

`STAT_IFMT = $f000`

File (stat (782) record) mode: File type bit mask

`STAT_IFREG = $8000`

File (stat (782) record) mode: Regular file

`STAT_IFSOCK = $c000`

File (stat (782) record) mode: Socket

`STAT_IRGRP = STAT_IROTH shl 3`

File (stat (782) record) mode: Group read permission

`STAT_IROTH = $4`

File (stat (782) record) mode: Other read permission

`STAT_IRUSR = STAT_IROTH shl 6`

File (stat (782) record) mode: Owner read permission

`STAT_IRWXG = STAT_IRW XO shl 3`

File (stat (782) record) mode: Group permission bits mask

`STAT_IRW XO = $7`

File (stat (782) record) mode: Other permission bits mask

`STAT_IRW XU = STAT_IRW XO shl 6`

File (stat (782) record) mode: Owner permission bits mask

`STAT_ISGID = $0400`

File (stat (782) record) mode: GID bit set

`STAT_ISUID = $0800`

File (stat (782) record) mode: UID bit set

`STAT_ISVTX = $0200`

File (stat (782) record) mode: Sticky bit set

`STAT_IWGRP = STAT_IWOTH shl 3`

File (stat (782) record) mode: Group write permission

`STAT_IWOTH = $2`

File (stat (782) record) mode: Other write permission

`STAT_IWUSR = STAT_IWOTH shl 6`

File (stat (782) record) mode: Owner write permission

`STAT_IXGRP = STAT_IXOTH shl 3`

File (stat (782) record) mode: Others execute permission

`STAT_IXOTH = $1`

File (stat (782) record) mode: Others execute permission

`STAT_IXUSR = STAT_IXOTH shl 6`

File (stat (782) record) mode: Others execute permission

`syscall_nr_access = 33`

```
syscall_nr_acct = 51

syscall_nr_adjtimex = 124

syscall_nr_afs_syscall = 137

syscall_nr_alarm = 27

syscall_nr_bdflush = 134

syscall_nr_break = 17

syscall_nr_brk = 45

syscall_nr_chdir = 12

syscall_nr_chmod = 15

syscall_nr_chown = 16

syscall_nr_chroot = 61

syscall_nr_clone = 120

syscall_nr_close = 6

syscall_nr_creat = 8

syscall_nr_create_module = 127

syscall_nr_delete_module = 129

syscall_nr_dup = 41

syscall_nr_dup2 = 63

syscall_nr_execve = 11
```

```
syscall_nr_exit = 1

syscall_nr_fchdir = 133

syscall_nr_fchmod = 94

syscall_nr_fchown = 95

syscall_nr_fcntl = 55

syscall_nr_fdatasync = 148

syscall_nr_flock = 143

syscall_nr_fork = 2

syscall_nr_fstat = 108

syscall_nr_fstatfs = 100

syscall_nr_fsync = 118

syscall_nr_ftime = 35

syscall_nr_ftruncate = 93

syscall_nr_getdents = 141

syscall_nr_getegid = 50

syscall_nr_geteuid = 49

syscall_nr_getgid = 47

syscall_nr_getgroups = 80

syscall_nr_getitimer = 105
```

syscall_nr_getpgid = 132

syscall_nr_getpgrp = 65

syscall_nr_getpid = 20

syscall_nr_getppid = 64

syscall_nr_getpriority = 96

syscall_nr_getresuid = 165

syscall_nr_getrlimit = 76

syscall_nr_getrusage = 77

syscall_nr_getsid = 147

syscall_nr_gettimeofday = 78

syscall_nr_getuid = 24

syscall_nr_get_kernel_syms = 130

syscall_nr_gtty = 32

syscall_nr_idle = 112

syscall_nr_init_module = 128

syscall_nr_ioctl = 54

syscall_nr_ioperm = 101

syscall_nr_iopl = 110

syscall_nr_ipc = 117

`syscall_nr_kill = 37`

`syscall_nr_link = 9`

`syscall_nr_lock = 53`

`syscall_nr_lseek = 19`

`syscall_nr_lstat = 107`

`syscall_nr_mkdir = 39`

`syscall_nr_mknod = 14`

`syscall_nr_mlock = 150`

`syscall_nr_mlockall = 152`

`syscall_nr_mmap = 90`

`syscall_nr_modify_ldt = 123`

`syscall_nr_mount = 21`

`syscall_nr_mprotect = 125`

`syscall_nr_mpx = 56`

`syscall_nr_mremap = 163`

`syscall_nr_msync = 144`

`syscall_nr_munlock = 151`

`syscall_nr_munlockall = 153`

`syscall_nr_munmap = 91`

syscall_nr_nanosleep = 162

syscall_nr_nice = 34

syscall_nr_oldfstat = 28

syscall_nr_oldlstat = 84

syscall_nr_oldolduname = 59

syscall_nr_oldstat = 18

syscall_nr_olduname = 109

syscall_nr_open = 5

syscall_nr_pause = 29

syscall_nr_personality = 136

syscall_nr_phys = 52

syscall_nr_pipe = 42

syscall_nr_poll = 168

syscall_nr_prof = 44

syscall_nr_profil = 98

syscall_nr_ptrace = 26

syscall_nr_query_module = 167

syscall_nr_quotactl = 131

syscall_nr_read = 3

`syscall_nr_readdir = 89`

`syscall_nr_readlink = 85`

`syscall_nr_readv = 145`

`syscall_nr_reboot = 88`

`syscall_nr_rename = 38`

`syscall_nr_rmdir = 40`

`syscall_nr_sched_getparam = 155`

`syscall_nr_sched_getscheduler = 157`

`syscall_nr_sched_get_priority_max = 159`

`syscall_nr_sched_get_priority_min = 160`

`syscall_nr_sched_rr_get_interval = 161`

`syscall_nr_sched_setparam = 154`

`syscall_nr_sched_setscheduler = 156`

`syscall_nr_sched_yield = 158`

`syscall_nr_select = 82`

`syscall_nr_setdomainname = 121`

`syscall_nr_setfsgid = 139`

`syscall_nr_setfsuid = 138`

`syscall_nr_setgid = 46`

`syscall_nr_setgroups = 81`

`syscall_nr_sethostname = 74`

`syscall_nr_setitimer = 104`

`syscall_nr_setpgid = 57`

`syscall_nr_setpriority = 97`

`syscall_nr_setregid = 71`

`syscall_nr_setresuid = 164`

`syscall_nr_setreuid = 70`

`syscall_nr_setrlimit = 75`

`syscall_nr_setsid = 66`

`syscall_nr_settimeofday = 79`

`syscall_nr_setuid = 23`

`syscall_nr_setup = 0`

`syscall_nr_sgetmask = 68`

`syscall_nr_sigaction = 67`

`syscall_nr_sigaltstack = 186`

`syscall_nr_signal = 48`

`syscall_nr_sigpending = 73`

`syscall_nr_sigprocmask = 126`

`syscall_nr_sigreturn = 119`

`syscall_nr_sigsuspend = 72`

`syscall_nr_socketcall = 102`

`syscall_nr_ssetmask = 69`

`syscall_nr_stat = 106`

`syscall_nr_statfs = 99`

`syscall_nr_stime = 25`

`syscall_nr_stty = 31`

`syscall_nr_swapoff = 115`

`syscall_nr_swapon = 87`

`syscall_nr_symlink = 83`

`syscall_nr_sync = 36`

`syscall_nr_sysfs = 135`

`syscall_nr_sysinfo = 116`

`syscall_nr_syslog = 103`

`syscall_nr_time = 13`

`syscall_nr_times = 43`

`syscall_nr_truncate = 92`

`syscall_nr_ulimit = 58`

syscall_nr_umask = 60

syscall_nr_umount = 22

syscall_nr_uname = 122

syscall_nr_unlink = 10

syscall_nr_uselib = 86

syscall_nr_ustat = 62

syscall_nr_utime = 30

syscall_nr_vhangup = 111

syscall_nr_vm86 = 166

syscall_nr_vm86old = 113

syscall_nr_wait4 = 114

syscall_nr_waitpid = 7

syscall_nr_write = 4

syscall_nr_writev = 146

syscall_nr__llseek = 140

syscall_nr__newselect = 142

syscall_nr__sysctl = 149

Sys_E2BIG = 7

Sys_EACCES = 13

Sys_EADDRINUSE = 98

Sys_EADDRNOTAVAIL = 99

Sys_EADV = 68

Sys_EAFNOSUPPORT = 97

Sys_EAGAIN = 11

Sys_EALREADY = 114

Sys_EBADE = 52

Sys_EBADF = 9

Sys_EBADFD = 77

Sys_EBADMSG = 74

Sys_EBADR = 53

Sys_EBADRQC = 56

Sys_EBADSLT = 57

Sys_EBFONT = 59

Sys_EBUSY = 16

Sys_ECHILD = 10

Sys_ECHRNG = 44

Sys_ECOMM = 70

Sys_ECONNABORTED = 103

Sys_ECONNREFUSED = 111

Sys_ECONNRESET = 104

Sys_EDEADLK = 35

Sys_EDEADLOCK = 58

Sys_EDESTADDRREQ = 89

Sys_EDOM = 33

Sys_EDOTDOT = 73

Sys_EDQUOT = 122

Sys_EEXIST = 17

Sys_EFAULT = 14

Sys_EFBIG = 27

Sys_EHOSTDOWN = 112

Sys_EHOSTUNREACH = 113

Sys_EIDRM = 43

Sys_EILSEQ = 84

Sys_EINPROGRESS = 115

Sys_EINTR = 4

Sys_EINVAL = 22

Sys_EIO = 5

Sys_EISCONN = 106

Sys_EISDIR = 21

Sys_EISNAM = 120

Sys_EL2HLT = 51

Sys_EL2NSYNC = 45

Sys_EL3HLT = 46

Sys_EL3RST = 47

Sys_ELIBACC = 79

Sys_ELIBBAD = 80

Sys_ELIBEXEC = 83

Sys_ELIBMAX = 82

Sys_ELIBSCN = 81

Sys_ELN RNG = 48

Sys_ELOOP = 40

Sys_EMFILE = 24

Sys_EMLINK = 31

Sys_MSGSIZE = 90

Sys_EMULTIHOP = 72

Sys_ENAMETOOLONG = 36

Sys_ENAVAIL = 119

Sys_ENETDOWN = 100

Sys_ENETRESET = 102

Sys_ENETUNREACH = 101

Sys_ENFILE = 23

Sys_ENOANO = 55

Sys_ENOBUFS = 105

Sys_ENOCSI = 50

Sys_ENODATA = 61

Sys_ENODEV = 19

Sys_ENOENT = 2

Sys_ENOEXEC = 8

Sys_ENOLCK = 37

Sys_ENOLINK = 67

Sys_ENOMEM = 12

Sys_ENOMSG = 42

Sys_ENONET = 64

Sys_ENOPKG = 65

Sys_ENOPROTOOPT = 92

Sys_ENOSPC = 28

Sys_ENOSR = 63

Sys_ENOSTR = 60

Sys_ENOSYS = 38

Sys_ENOTBLK = 15

Sys_ENOTCONN = 107

Sys_ENOTDIR = 20

Sys_ENOTEMPTY = 39

Sys_ENOTNAM = 118

Sys_ENOTSOCK = 88

Sys_ENOTTY = 25

Sys_ENOTUNIQ = 76

Sys_ENXIO = 6

Sys_EOPNOTSUPP = 95

Sys_EOVERFLOW = 75

Sys_EPERM = 1

Sys_EPFNOSUPPORT = 96

Sys_EPIPE = 32

Sys_EPROTO = 71

Sys_EPROTONOSUPPORT = 93

Sys_EPROTOTYPE = 91

Sys_ERANGE = 34

Sys_EREMCHG = 78

Sys_EREMOTE = 66

Sys_EREMOTEIO = 121

Sys_ERESTART = 85

Sys_EROFS = 30

Sys_ERROR_MAX = \$fff

Sys_ESHUTDOWN = 108

Sys_ESOCKTNOSUPPORT = 94

Sys_ESPIPE = 29

Sys_ESRCH = 3

Sys_ESRMNT = 69

Sys_ESTALE = 116

Sys ESTRPIPE = 86

Sys_ETIME = 62

Sys_ETIMEDOUT = 110

Sys_ETOOMANYREFS = 109

Sys_ETXTBSY = 26

Sys_EUCLEAN = 117

Sys_EUNATCH = 49

Sys_EUSERS = 87

Sys_EWOULDBLOCK = Sys_EAGAIN

Sys_EXDEV = 18

Sys_EXFULL = 54

TAB0 = \$00000000

TAB1 = \$00008000

TAB2 = \$00010000

TAB3 = \$00018000

TABDLY = \$00018000

TCFLSH = \$540B

TCGETA = \$5405

TCGETS = \$5401

TCIFLUSH = 0

TCIOFF = 2

TCIOFLUSH = 2

TCION = 3

TCOFLUSH = 1

TCOOFF = 0

TCOON = 1

TCSADRAIN = 1

TCSAFLUSH = 2

TCSANOW = 0

TCSBRK = \$5409

TCSBRKP = \$5425

TCSETA = \$5406

TCSETAF = \$5408

TCSETAW = \$5407

TCSETS = \$5402

TCSETSF = \$5404

TCSETSW = \$5403

TCXONC = \$540A

TIOCCONS = \$541D

TIOCEXCL = \$540C

TIOCGETD = \$5424

TIOCGICOUNT = \$545D

TIOCGLOCKTRMIO = \$5456

TIOCGPGRP = \$540F

TIOCGSERIAL = \$541E

TIOCGSOFTCAR = \$5419

TIOCGWINSZ = \$5413

TIOCINQ = FIONREAD

TIOCLINUX = \$541C

TIOCMBIC = \$5417

TIOCMBIS = \$5416

TIOCMGET = \$5415

TIOCMWAIT = \$545C

TIOCMSET = \$5418

TIOCM_CAR = \$040

TIOCM_CD = TIOCM_CAR

TIOCM_CTS = \$020

TIOCM_DSR = \$100

TIOCM_DTR = \$002

TIOCM_LE = \$001

TIOCM_OUT1 = \$2000

TIOCM_OUT2 = \$4000

TIOCM_RI = TIOCM_RNG

TIOCM_RNG = \$080

TIOCM_RTS = \$004

TIOCM_SR = \$010

TIOCM_ST = \$008

TIOCNOTTY = \$5422

TIOCNXCL = \$540D

TIOCOUTQ = \$5411

TIOCPKT = \$5420

TIOCPKT_DATA = 0

TIOCPKT_DOSTOP = 32

TIOCPKT_FLUSHREAD = 1

TIOCPKT_FLUSHWRITE = 2

TIOCPKT_NOSTOP = 16

TIOCPKT_START = 8

TIOCPKT_STOP = 4

TIOCSCTTY = \$540E

TIOCSERCONFIG = \$5453

TIOCSERGETLSR = \$5459

TIOCSERGETMULTI = \$545A

TIOCSERGSTRUCT = \$5458

TIOCSERGWILD = \$5454

TIOCSERSETMULTI = \$545B

TIOCSERSWILD = \$5455

TIOCSETD = \$5423

TIOCSLCKTRMIOS = \$5457

TIOCSPGRP = \$5410

TIOCSSERIAL = \$541F

TIOCSSOFTCAR = \$541A

TIOCSTI = \$5412

TIOCSWINSZ = \$5414

TIOCTTYGSTRUCT = \$5426

TOSTOP = \$0000100

VDISCARD = 13

VEOF = 4

VEOL = 11

VEOL2 = 16

VERASE = 2

VINTR = 0

VKILL = 3

VLNEXT = 15

VMIN = 6

VQUIT = 1

VREPRINT = 12

VSTART = 8

VSTOP = 9

VSUSP = 10

VSWTC = 7

VT0 = \$00000000

VT1 = \$00040000

VTDLY = \$00040000

VTIME = 5

VWERASE = 14

Wait_Any = -1

WaitPID (860): Wait on any process

Wait_Clone = \$80000000

WaitPID (860): Wait on clone processes only.

Wait_MyPGRP = 0

WaitPID (860): Wait processes from current process group

Wait_NoHang = 1

WaitPID (860): Do not wait

Wait_UnTraced = 2

WaitPID (860): Also report stopped but untraced processes

WNOHANG = \$1

Waitpid (860) option: Do not wait for processes to terminate.

WUNTRACED = \$2

Waitpid (860) option: Also report children which were stopped but not yet reported

W_OK = 2

Access (787) call test: write allowed

XCASE = \$00000004

XTABS = \$0001800

X_OK = 1

Access (787) call test: execute allowed

__WCLONE = \$800000000

Waitpid option: Wait for clone children only

Types

ComStr =

Command-line string type.

dev_t = Word

Device descriptor type

```
dirent = packed record
  ino : LongInt;
  off : LongInt;
  reclen : Word;
  name : Array[0..255] of Char;
end
```

Record used in the ReadDir (839) function to return files in a directory.

DirStr =

Filename directory part string type.

ExtStr =

Filename extension part string type.

fdSet = Array[0..7] of LongInt

Array containing file descriptor bitmask for the Select (841) call.

NameStr =

Filename name part string type.

PathStr =

Filename path part string type.

PDir = ^TDir

Pointer to TDir (783) record

pdirent = ^dirent

Pointer to Dirent (779) record.

pfdsset = ^fdSet

Pointer to FDSet (811) array.

pfpstate = ^tfpstate

Pointer to tfpstate (784) record.

pglob = ^tglob

Pointer to TGlob (784) record.

PSigActionRec = ^SigActionRec

Pointer to SigActionRec (781) record.

PSigAltStack = ^SigAltStack

Pointer to SigAltStack (781) record

PSigContextRec = ^SigContextRec

Pointer to SigContextRec (781) record

PSignalHandler = ^SignalHandler

Pointer to SignalHandler (781) type.

PSignalRestorer = ^SignalRestorer

Pointer to SignalRestorer (781) type

PSigSet = ^SigSet

Pointer to signal set.

pstack_t = ^stack_t

Pointer to stack_t (782) record

PStat = ^Stat

Pointer to Stat (782) record.

PStatFS = ^Statfs

Pointer to StatFS (782) record.

PSysCallRegs = ^SysCallRegs

Pointer to SysCallRegs (783) record.

PSysInfo = ^TSysinfo

Pointer to TSysInfo (785) record.

ptimeval = ^timeval

Pointer to TTimeVal (786) record

ptimezone = ^timezone

Pointer to TimeZone (785) record.

PUTimeBuf = ^UTimeBuf

Pointer to UTimeBuf (786) record

PUTSName = ^utsname

Pointer to UTSName (786) record.

```
SigActionRec = packed record
  Handler : record
  end;
  Sa_Mask : SigSet;
  Sa_Flags : LongInt;
  Sa_restorer : SignalRestorer;
end
```

Record used in SigAction (844) call.

```
SigAltStack = record
  ss_sp : pointer;
  ss_flags : LongInt;
  ss_size : Size_T;
end
```

Alternate stack registers record

```
SigContextRec = record
  gs : Word;
  __gsh : Word;
  fs : Word;
  __fsh : Word;
  es : Word;
  __esh : Word;
  ds : Word;
  __dsh : Word;
  edi : cardinal;
  esi : cardinal;
  ebp : cardinal;
  esp : cardinal;
  ebx : cardinal;
  edx : cardinal;
  ecx : cardinal;
  eax : cardinal;
  trapno : cardinal;
  err : cardinal;
  eip : cardinal;
  cs : Word;
  __csh : Word;
  eflags : cardinal;
  esp_at_signal : cardinal;
  ss : Word;
  __ssh : Word;
  fpstate : pfpstate;
  oldmask : cardinal;
  cr2 : cardinal;
end
```

The above records contain information about the processor state and process state at the moment a signal is sent to your program.

```
SignalHandler = procedure(Sig: LongInt)
```

Function prototype for the Signal (846) call.

```
SignalRestorer = procedure
```

Signal restorer function prototype

```
SigSet = LongInt
```

Signal set type

Size_T = cardinal

Size type

stack_t = SigAltStack

Alias for SigAltStack (781) type

Stat = packed record

```

    dev : dev_t;
    pad1 : Word;
    ino : LongInt;
    mode : Word;
    nlink : Word;
    uid : Word;
    gid : Word;
    rdev : dev_t;
    pad2 : Word;
    size : LongInt;
    blksize : LongInt;
    blocks : LongInt;
    atime : LongInt;
    unused1 : LongInt;
    mtime : LongInt;
    unused2 : LongInt;
    ctime : LongInt;
    unused3 : LongInt;
    unused4 : LongInt;
    unused5 : LongInt;
end

```

Record describing an inode (file) in the fstat (816) call.

Statfs = packed record

```

    fstype : LongInt;
    bsize : LongInt;
    blocks : LongInt;
    bfree : LongInt;
    bavail : LongInt;
    files : LongInt;
    ffree : LongInt;
    fsid : LongInt;
    namelen : LongInt;
    spare : Array[0..6] of LongInt;
end

```

Record describing a file system in the fsstat (815) call.

SysCallRegs = record

```

    reg1 : LongInt;

```

```

    reg2 : LongInt;
    reg3 : LongInt;
    reg4 : LongInt;
    reg5 : LongInt;
    reg6 : LongInt;
end

```

Register describing system calls.

```
TCloneFunc = function(args: pointer) : LongInt
```

Clone function prototype.

```

TDir = packed record
    fd : Integer;
    loc : LongInt;
    size : Integer;
    buf : pdirent;
    nextoff : LongInt;
    dd_max : Integer;
    lock : pointer;
end

```

Record used in [OpenDir \(836\)](#) and [ReadDir \(839\)](#) calls

```
TDirEnt = dirent
```

Alias for [DirEnt \(779\)](#) record

```

Termio = packed record
    c_iflag : Word;
    c_oflag : Word;
    c_cflag : Word;
    c_lflag : Word;
    c_line : Word;
    c_cc : Array[0..NCC-1] of Char;
end

```

Terminal I/O description record (small)

```

Termios = record
    c_iflag : Cardinal;
    c_oflag : Cardinal;
    c_cflag : Cardinal;
    c_lflag : Cardinal;
    c_line : Char;
    c_cc : Array[0..NCCS-1] of Byte;
    c_ispeed : LongInt;
    c_ospeed : LongInt;
end

```

Terminal I/O description record

```
TFDSet = fdSet
```

Alias for FDSets (811) type.

```
tfpreg = record
  significand : Array[0..3] of Word;
  exponent   : Word;
end
```

Record describing floating point register in signal handler.

```
tfpstate = record
  cw : cardinal;
  sw : cardinal;
  tag : cardinal;
  ipoff : cardinal;
  cssel : cardinal;
  dataoff : cardinal;
  datasel : cardinal;
  st : Array[0..7] of tfpreg;
  status : cardinal;
end
```

Record describing floating point unit in signal handler.

```
tglob = record
  name : pchar;
  next : pglob;
end
```

Record containing one entry in the result of Glob (826)

```
timespec = packed record
  tv_sec : LongInt;
  tv_nsec : LongInt;
end
```

Time interval for the NanoSleep (834) function.

```
timeval = packed record
  sec : LongInt;
  usec : LongInt;
end
```

Record specifying a time interval.

```
timezone = packed record
```

```

    minuteswest : LongInt;
    dsttime : LongInt;
end

```

Record describing a timezone

```

tmmapargs = record
    address : LongInt;
    size : LongInt;
    prot : LongInt;
    flags : LongInt;
    fd : LongInt;
    offset : LongInt;
end

```

Record containing mmap args.

```

Tpipe = Array[1..2] of LongInt

```

Array describing a pipe pair of filedescriptors.

```

TSigAction = procedure(Sig: LongInt; SigContext: SigContextRec)

```

Function prototype for SigAction ([844](#)) call.

```

TStat = Stat

```

Alias for Stat ([782](#)) record.

```

TStatFS = Statfs

```

Alias for StatFS ([782](#)) type.

```

TSysCallRegs = SysCallRegs

```

Alias for SysCallRegs ([783](#)) record

```

TSysinfo = packed record
    uptime : LongInt;
    loads : Array[1..3] of LongInt;
    totalram : LongInt;
    freeram : LongInt;
    sharedram : LongInt;
    bufferram : LongInt;
    totalswap : LongInt;
    freeswap : LongInt;
    procs : Integer;
    s : String;
end

```

Record with system information, used by the SysInfo ([851](#)) call.

`TTermio = Termio`

Alias for `TermIO` (783) record

`TTermios = Termios`

Alias for `Termios` (784) record.

`TTimeVal = timeval`

Alias for `TimeVal` (784) record.

`TTimeZone = timezone`

Alias for `TimeZone` (785) record.

`TUTimeBuf = UTimeBuf`

Alias for `UTimBuf` (786) record.

`TUTSName = utsname`

Alias for `UTSName` (786) record.

`TWinSize = winsize`

Alias for `WinSize` (787) record.

```
UTimBuf = packed record
  actime : LongInt;
  modtime : LongInt;
end
```

Record used in `Utime` (859) to set file access and modification times.

`UTimeBuf = UTimBuf`

Alias for `UTimBuf` (786) record.

```
utsname = packed record
  sysname : Array[0..64] of Char;
  nodename : Array[0..64] of Char;
  release : Array[0..64] of Char;
  version : Array[0..64] of Char;
  machine : Array[0..64] of Char;
  domainname : Array[0..64] of Char;
end
```

The elements of this record are null-terminated C style strings, you cannot access them directly.

```
winsize = packed record
  ws_row : Word;
  ws_col : Word;
  ws_xpixel : Word;
  ws_ypixel : Word;
end
```

Record describing terminal window size.

Variables

`ErrNo` : `LongInt`

Error number of last operation.

`LinuxError` : `LongInt`

`Linuxerror` is the variable in which the procedures in the linux unit report errors.

`tzdaylight` : `Boolean`

Indicates whether daylight savings time is active.

`tzname` : `Array[boolean]` of `pchar`

Timezone name.

`tzseconds` : `LongInt`

Seconds west of GMT

22.12 Procedures and functions

Access

Synopsis: Check file access

Declaration: `function Access(Path: PathStr; mode: Integer) : Boolean`

Visibility: default

Description: `Access` tests user's access rights on the specified file. Mode is a mask existing of one or more of the following:

R_OKUser has read rights.

W_OKUser has write rights.

X_OKUser has execute rights.

F_OKFile exists.

The test is done with the real user ID, instead of the effective user ID. If access is denied, or an error occurred, `False` is returned.

Errors: `LinuxError` is used to report errors:

sys_eaccessThe requested access is denied, either to the file or one of the directories in its path.

sys_einvalMode was incorrect.

sys_enoentA directory component in `Path` doesn't exist or is a dangling symbolic link.

sys_enotdirA directory component in `Path` is not a directory.

sys_enomemInsufficient kernel memory.

`sys_eLOOPPath` has a circular symbolic link.

See also: `Chown` ([794](#)), `Chmod` ([792](#))

Listing: `./olinuxex/ex26.pp`

Program `Example26`;

{ Program to demonstrate the Access function. }

Uses `oldlinux`;

```
begin
  if Access ( '/etc/passwd',W_OK) then
    begin
      Writeln ( 'Better check your system.' );
      Writeln ( 'I can write to the /etc/passwd file !' );
    end;
  end.
```

Alarm

Synopsis: Schedule an alarm signal to be delivered

Declaration: `function Alarm(Sec: LongInt) : LongInt`

Visibility: `default`

Description: `Alarm` schedules an alarm signal to be delivered to your process in `Sec` seconds. When `Sec` seconds have elapsed, Linux will send a `SIGALRM` signal to the current process. If `Sec` is zero, then no new alarm will be set. Whatever the value of `Sec`, any previous alarm is cancelled.

The function returns the number of seconds till the previously scheduled alarm was due to be delivered, or zero if there was none.

See also: `SigAction` ([844](#))

Listing: `./olinuxex/ex59.pp`

Program `Example59`;

{ Program to demonstrate the Alarm function. }

Uses `oldlinux`;

Procedure `AlarmHandler(Sig : longint); cdecl`;

```
begin
  Writeln ( 'Got to alarm handler' );
end;

begin
  Writeln ( 'Setting alarm handler' );
  Signal(SIGALRM, @AlarmHandler);
  Writeln ( 'Scheduling Alarm in 10 seconds' );
  Alarm(10);
  Writeln ( 'Pausing' );
  Pause;
```

```

    Writeln ( 'Pause returned ' );
end.

```

AssignPipe

Synopsis: Create a set of pipe file handlers

Declaration: `function AssignPipe(var pipe_in: LongInt; var pipe_out: LongInt) : Boolean`
`function AssignPipe(var pipe_in: text; var pipe_out: text) : Boolean`
`function AssignPipe(var pipe_in: file; var pipe_out: file) : Boolean`

Visibility: default

Description: AssignPipe creates a pipe, i.e. two file objects, one for input, one for output. What is written to Pipe_out, can be read from Pipe_in.

This call is overloaded. The in and out pipe can take three forms: an typed or untyped file, a text file or a file descriptor.

If a text file is passed then reading and writing from/to the pipe can be done through the usual `Readln(Pipe_in, ...)` and `Writeln(Pipe_out, ...)` procedures.

The function returns `True` if everything went successfully, `False` otherwise.

Errors: In case the function fails and returns `False`, `LinuxError` is used to report errors:

`sys_enfile` Too many file descriptors for this process.

`sys_enfile` The system file table is full.

See also: `POpen` (838), `MkFifo` (832)

Listing: `./olinuxex/ex36.pp`

Program Example36;

{ Program to demonstrate the AssignPipe function. }

Uses oldlinux;

Var pipi, pipo : Text;
 s : **String**;

begin

```

    Writeln ( 'Assigning Pipes.' );
    If Not assignpipe(pipi, pipo) then
        Writeln('Error assigning pipes !', LinuxError);
    Writeln ( 'Writing to pipe, and flushing.' );
    Writeln ( pipo, 'This is a textstring' ); close(pipo);
    Writeln ( 'Reading from pipe.' );
    While not eof(pipi) do
        begin
            Readln ( pipi, s );
            Writeln ( 'Read from pipe : ', s );
        end;
    close ( pipi );
    writeln ( 'Closed pipes.' );
    writeln

```

end.

AssignStream

Synopsis: Assign stream for in and output to a program

Declaration:

```
function AssignStream(var StreamIn: text;var Streamout: text;
                    const Prog: String) : LongInt
function AssignStream(var StreamIn: Text;var StreamOut: Text;
                    var StreamErr: Text;const prog: String) : LongInt
```

Visibility: default

Description: AssignStream creates a 2 or 3 pipes, i.e. two (or three) file objects, one for input, one for output, (and one for standard error) the other ends of these pipes are connected to standard input and output (and standard error) of Prog. Prog is the name of a program (including path) with options, which will be executed.

What is written to StreamOut, will go to the standard input of Prog. Whatever is written by Prog to it's standard output can be read from StreamIn. Whatever is written by Prog to it's standard error read from StreamErr, if present.

Reading and writing happens through the usual Readln(StreamIn,...) and Writeln(StreamOut,...) procedures.

Remark: You should *not* use Reset or Rewrite on a file opened with POpen. This will close the file before re-opening it again, thereby closing the connection with the program.

The function returns the process ID of the spawned process, or -1 in case of error.

Errors: In case of error (return value -1) LinuxError is used to report errors:

sys_enfile Too many file descriptors for this process.

sys_enfile The system file table is full.

Other errors include the ones by the fork and exec programs

See also: AssignPipe ([789](#)), POpen ([838](#))

Listing: ./olinuxex/ex38.pp

Program Example38;

{ Program to demonstrate the AssignStream function. }

Uses oldlinux;

Var Si, So : Text;
 S : **String**;
 i : longint;

begin
 if not (paramstr(1) = '-son') **then**
 begin
 Writeln ('Calling son');
 Assignstream (Si, So, './ex38 -son');
 if linuxerror <> 0 **then**
 begin
 writeln ('AssignStream failed !');
 halt(1);
 end;
 Writeln ('Speaking to son');
 For i:=1 **to** 10 **do**

```

begin
  writeln (so, 'Hello son ! ');
  if ioreult<>0 then writeln ( 'Can''t speak to son... ');
end;
For i:=1 to 3 do writeln (so, 'Hello chap ! ');
close (so);
while not eof(si) do
begin
  readln (si,s);
  writeln ( 'Father: Son said : ',S);
end;
Writeln ( 'Stopped conversation ');
Close (Si);
Writeln ( 'Put down phone ');
end
Else
begin
  Writeln ( 'This is the son ');
  While not eof (input) do
  begin
    readln (s);
    if pos ( 'Hello son ! ',S)<>0 then
      Writeln ( 'Hello Dad ! ')
    else
      writeln ( 'Who are you ? ');
    end;
  close (output);
end
end.

```

Basename

Synopsis: Return basename of a file

Declaration: `function Basename(const path: PathStr;const suf: PathStr) : PathStr`

Visibility: default

Description: Returns the filename part of Path, stripping off Suf if it exists. The filename part is the whole name if Path contains no slash, or the part of Path after the last slash. The last character of the result is not a slash, unless the directory is the root directory.

Errors: None.

See also: [DirName \(798\)](#), [FExpand \(811\)](#)

Listing: ./olinuxex/ex48.pp

Program Example48;

{ Program to demonstrate the BaseName function. }

Uses oldlinux;

Var S : **String**;

begin
 S:=FExpand(**Paramstr**(0));

```
WriteIn ( 'This program is called : ', Basename(S, ''));  
end.
```

CFMakeRaw

Synopsis: Sets flags in Termios (784) record.

Declaration: `procedure CFMakeRaw(var tios: Termios)`

Visibility: default

Description: CFMakeRaw sets the flags in the Termios structure Tios to a state so that the terminal will function in Raw Mode.

For an example, see TCGetAttr (855).

Errors: None.

See also: CFSetOSpeed (792), CFSetISpeed (792)

CFSetISpeed

Synopsis: Set input baud rate in Termios (784) record

Declaration: `procedure CFSetISpeed(var tios: Termios; speed: Cardinal)`

Visibility: default

Description: CFSetISpeed Sets the input baudrate in the TermIOS structure Tios to Speed.

Errors: None.

See also: CFSetOSpeed (792), CFMakeRaw (792)

CFSetOSpeed

Synopsis: Set output baud rate in Termios (784) record

Declaration: `procedure CFSetOSpeed(var tios: Termios; speed: Cardinal)`

Visibility: default

Description: CFSetOSpeed Sets the output baudrate in the Termios structure Tios to Speed.

Errors: None.

See also: CFSetISpeed (792), CFMakeRaw (792)

Chmod

Synopsis: Change file permission bits

Declaration: `function Chmod(path: PathStr; Newmode: LongInt) : Boolean`

Visibility: default

Description: Chmod Sets the Mode bits of the file in Path to NewMode. Newmode can be specified by 'or'-ing the following:

S_ISUIDSet user ID on execution.
S_ISGIDSet Group ID on execution.
S_ISVTXSet sticky bit.
S_IRUSRRead by owner.
S_IWUSRWrite by owner.
S_IXUSRExecute by owner.
S_IRGRPRead by group.
S_IWGRPWrite by group.
S_IXGRPExecute by group.
S_IROTHRead by others.
S_IWOTHWrite by others.
S_IXOTHExecute by others.
S_IRWXORead, write, execute by others.
S_IRWXGRead, write, execute by groups.
S_IRWXURead, write, execute by user.

Errors: Errors are returned in `LinuxError`.

sys_epermThe effective UID doesn't match the ownership of the file, and is not zero. Owner or group were not specified correctly.
sys_eaccessOne of the directories in `Path` has no search (=execute) permission.
sys_enoentA directory entry in `Path` does not exist or is a symbolic link pointing to a non-existent directory.
sys_enomemInsufficient kernel memory.
sys_erofsThe file is on a read-only filesystem.
sys_eloop`Path` has a reference to a circular symbolic link, i.e. a symbolic link, whose expansion points to itself.

See also: `Chown` ([794](#)), `Access` ([787](#)), `Octal` ([836](#))

Listing: `./olinuxex/ex23.pp`

Program `Example23`;

{ Program to demonstrate the Chmod function. }

Uses `oldlinux`;

Var `F` : `Text`;

```
begin
  { Create a file }
  Assign (f, 'testex21');
  Rewrite (F);
  Writeln (f, '#!/bin/sh');
  Writeln (f, 'echo Some text for this file');
  Close (F);
  { Octal() makes the correct number from a
    number that LOOKS octal }
  Chmod ('testex21', octal (777));
  { File is now executable }
  execl ('./testex21');
end.
```

Chown

Synopsis: Change owner of file

Declaration: `function Chown(path: PathStr;NewUid: LongInt;NewGid: LongInt) : Boolean`

Visibility: default

Description: Chown sets the User ID and Group ID of the file in Path to NewUid, NewGid The function returns True if the call was succesfull, False if the call failed.

Errors: Errors are returned in `LinuxError`.

sys_epermThe effective UID doesn't match the ownership of the file, and is not zero. Owner or group were not specified correctly.

sys_eaccessOne of the directories in Path has no search (=execute) permission.

sys_enoentA directory entry in Path does not exist or is a symbolic link pointing to a non-existent directory.

sys_enomemInsufficient kernel memory.

sys_erofsThe file is on a read-only filesystem.

sys_eloopPath has a reference to a circular symbolic link, i.e. a symbolic link, whose expansion points to itself.

See also: Chmod ([792](#)), Access ([787](#))

Listing: `./olinuxex/ex24.pp`

Program Example24;

{ Program to demonstrate the Chown function. }

Uses oldlinux;

Var UID,GID : Longint;
F : Text;

begin

```

    Writeln ('This will only work if you are root. ');
    Write ('Enter a UID : ');readln(UID);
    Write ('Enter a GID : ');readln(GID);
    Assign (f, 'test.txt');
    Rewrite (f);
    Writeln (f, 'The owner of this file should become : ');
    Writeln (f, 'UID : ',UID);
    Writeln (f, 'GID : ',GID);
    Close (F);
    if not Chown ('test.txt',UID,GID) then
        if LinuxError=Sys_EPERM then
            Writeln ('You are not root !')
        else
            Writeln ('Chmod failed with exit code : ',LinuxError)
        else
            Writeln ('Changed owner successfully !');
end.
```

Clone

Synopsis: Clone current process (create new thread)

Declaration: `function Clone(func: TCloneFunc; sp: pointer; flags: LongInt;
args: pointer) : LongInt`

Visibility: default

Description: `Clone` creates a child process which is a copy of the parent process, just like `Fork` (813) does. In difference with `Fork`, however, the child process shares some parts of it's execution context with its parent, so it is suitable for the implementation of threads: many instances of a program that share the same memory.

When the child process is created, it starts executing the function `Func`, and passes it `Args`. The return value of `Func` is either the explicit return value of the function, or the exit code of the child process.

The `sp` pointer points to the memory reserved as stack space for the child process. This address should be the top of the memory block to be used as stack.

The `Flags` determine the behaviour of the `Clone` call. The low byte of the `Flags` contains the number of the signal that will be sent to the parent when the child dies. This may be bitwise OR'ed with the following constants:

CLONE_VMParent and child share the same memory space, including memory (un)mapped with subsequent `mmap` calls.

CLONE_FSParent and child have the same view of the filesystem; the `chroot`, `chdir` and `umask` calls affect both processes.

CLONE_FILESthe file descriptor table of parent and child is shared.

CLONE_SIGHANDthe parent and child share the same table of signal handlers. The signal masks are different, though.

CLONE_PIDParent and child have the same process ID.

`Clone` returns the process ID in the parent process, and -1 if an error occurred.

Errors: On error, -1 is returned to the parent, and no child is created.

sys_eagainToo many processes are running.

sys_enomemNot enough memory to create child process.

See also: `Fork` (813)

Listing: `./olinuxex/ex71.pp`

```
program TestC { lone };
```

```
uses
```

```
  oldlinux , Errors , crt ;
```

```
const
```

```
  Ready : Boolean = false ;
```

```
  aChar : Char    = 'a' ;
```

```
function CloneProc ( Arg : Pointer ) : LongInt ; Cdecl ;
```

```
begin
```

```
  WriteLn ( 'Hello from the clone ' , PChar ( Arg ) ) ;
```

```
  repeat
```



```

    Write(aChar);
    Select(0,Nil,Nil,Nil,Nil);
until Ready;
WriteLn( 'Clone finished.' );
CloneProc := 1;
end;

var
    PID : LongInt;

procedure MainProc;
begin
    WriteLn( 'cloned process PID: ', PID );
    WriteLn( 'Press <ESC> to kill ... ' );
    repeat
        Write( '.' );
        Select(0,Nil,Nil,Nil,Nil);
        if KeyPressed then
            case ReadKey of
                #27: Ready := true;
                'a': aChar := 'A';
                'A': aChar := 'a';
                'b': aChar := 'b';
                'B': aChar := 'B';
            end;
        until Ready;
        WriteLn( 'Ready.' );
    end;

const
    StackSize = 16384;
    theFlags = CLONE_VM+CLONE_FS+CLONE_FILES+CLONE_SIGHAND;
    aMsg      : PChar = 'Oops !';

var
    theStack : Pointer;
    ExitStat : LongInt;

begin
    GetMem(theStack, StackSize);
    PID := Clone( @CloneProc,
                  Pointer( LongInt(theStack)+StackSize ),
                  theFlags,
                  aMsg);
    if PID < 0 then
        WriteLn( 'Error : ', LinuxError, ' when cloning.' )
    else
        begin
            MainProc;
            case WaitPID(0, @ExitStat, Wait_Untraced or wait_clone) of
                -1: WriteLn( 'error: ', LinuxError, '; ', StrError(LinuxError));
                0: WriteLn( 'error: ', LinuxError, '; ', StrError(LinuxError));
            else
                WriteLn( 'Clone exited with: ', ExitStat shr 8);
            end;
        end;
    FreeMem( theStack, StackSize );
end.

```

CloseDir

Synopsis: Close directory file descriptor

Declaration: `function CloseDir(p: PDir) : Integer`

Visibility: default

Description: `CloseDir` closes the directory pointed to by `p`. It returns zero if the directory was closed successfully, -1 otherwise.

For an example, see `OpenDir` ([836](#)).

Errors: Errors are returned in `LinuxError`.

See also: `OpenDir` ([836](#)), `ReadDir` ([839](#)), `SeekDir` ([841](#)), `TellDir` ([857](#))

CreateShellArgV

Synopsis: Create an array of null-terminated strings

Declaration: `function CreateShellArgV(const prog: String) : ppchar`
`function CreateShellArgV(const prog: Ansistring) : ppchar`

Visibility: default

Description: `CreateShellArgV` creates an array of 3 `PChar` pointers that can be used as arguments to `ExecVE` the first elements in the array will contain `/bin/sh`, the second will contain `-c`, and the third will contain `prog`.

The function returns a pointer to this array, of type `PPChar`.

Errors: None.

See also: `Shell` ([844](#))

Listing: `./olinuxex/ex61.pp`

```

Program ex61;

{ Example program to demonstrate the CreateShellArgV function }

uses oldlinux;

Var
  S: String;
  PP : PPchar;
  I : longint;

begin
  S:= 'script -a -b -c -d -e fghijk';
  PP:=CreateShellArgV(S);
  I:=0;
  If PP<>Nil then
    While PP[I]<>Nil do
      begin
        WriteLn ( 'Got : ',PP[I], ' ');
        Inc(I);
      end;
  end.
```

Dirname

Synopsis: Extract directory part from filename

Declaration: `function Dirname(const path: PathStr) : PathStr`

Visibility: default

Description: Returns the directory part of `Path`. The directory is the part of `Path` before the last slash, or empty if there is no slash. The last character of the result is not a slash, unless the directory is the root directory.

Errors: None.

See also: `BaseName` ([791](#)), `FExpand` ([811](#))

Listing: `./olinuxex/ex47.pp`

Program `Example47;`

{ Program to demonstrate the DirName function. }

Uses `oldlinux;`

Var `S : String;`

begin

`S:=FExpand(Paramstr(0));`

`WriteLn ('This program is in directory : ',Dirname(S));`

end.

Dup

Synopsis: Duplicate a file handle

Declaration: `function Dup(oldfile: LongInt;var newfile: LongInt) : Boolean`
`function Dup(var oldfile: text;var newfile: text) : Boolean`
`function Dup(var oldfile: file;var newfile: file) : Boolean`

Visibility: default

Description: Makes `NewFile` an exact copy of `OldFile`, after having flushed the buffer of `OldFile` in case it is a `Text` file or untyped file. Due to the buffering mechanism of Pascal, this has not the same functionality as the `dup` call in C. The internal Pascal buffers are not the same after this call, but when the buffers are flushed (e.g. after output), the output is sent to the same file. Doing an `lseek` will, however, work as in C, i.e. doing an `lseek` will change the fileposition in both files.

The function returns `False` in case of an error, `True` if successful.

Errors: In case of errors, `Linuxerror` is used to report errors.

`sys_ebadf``OldFile` hasn't been assigned.

`sys_emfile`Maximum number of open files for the process is reached.

See also: `Dup2` ([799](#))

Listing: `./olinuxex/ex31.pp`

```

program Example31;

{ Program to demonstrate the Dup function. }

uses oldlinux;

var f : text;

begin
  if not dup (output,f) then
    Writeln ( 'Dup Failed ! ');
  writeln ( 'This is written to stdout.' );
  writeln ( f, 'This is written to the dup file , and flushed' ); flush ( f );
  writeln
end .

```

Dup2

Synopsis: Duplicate one filehandle to another

Declaration: `function Dup2(oldfile: LongInt; newfile: LongInt) : Boolean`
`function Dup2(var oldfile: text; var newfile: text) : Boolean`
`function Dup2(var oldfile: file; var newfile: file) : Boolean`

Visibility: default

Description: Makes NewFile an exact copy of OldFile, after having flushed the buffer of OldFile in the case of text or untyped files.

NewFile can be an assigned file. If newfile was open, it is closed first. Due to the buffering mechanism of Pascal, this has not the same functionality as the dup2 call in C. The internal Pascal buffers are not the same after this call, but when the buffers are flushed (e.g. after output), the output is sent to the same file. Doing an lseek will, however, work as in C, i.e. doing a lseek will change the fileposition in both files.

The function returns True if succesful, false otherwise.

Errors: In case of error, Linuxerror is used to report errors.

sys_ebadf OldFile hasn't been assigned.

sys_emfile Maximum number of open files for the process is reached.

See also: Dup ([798](#))

Listing: ./olinuxex/ex32.pp

```

program Example31;

{ Program to demonstrate the Dup function. }

uses oldlinux;

var f : text;
    i : longint;

begin
  Assign (f, 'text.txt');

```

```

Rewrite (F);
For i:=1 to 10 do writeln (F,'Line : ',i);
if not dup2 (output,f) then
  Writeln ('Dup2 Failed !');
writeln ('This is written to stdout. ');
writeln (f,'This is written to the dup file , and flushed');
flush(f);
writeln;
{ Remove file . Comment this if you want to check flushing.}
Unlink ('text.txt');
end.

```

EpochToLocal

Synopsis: Convert epoch time to local time

Declaration: `procedure EpochToLocal(epoch: LongInt;var year: Word;var month: Word;
var day: Word;var hour: Word;var minute: Word;
var second: Word)`

Visibility: default

Description: Converts the epoch time (=Number of seconds since 00:00:00 , January 1, 1970, corrected for your time zone) to local date and time.

This function takes into account the timzeone settings of your system.

Errors: None

See also: [GetEpochTime \(820\)](#), [LocalToEpoch \(830\)](#), [GetTime \(824\)](#), [GetDate \(817\)](#)

Listing: ./olinuxex/ex3.pp

Program Example3;

{ Program to demonstrate the EpochToLocal function. }

Uses oldlinux;

Var Year,month,day,hour,minute,seconds : Word;

begin

EpochToLocal (GetEpochTime,Year,month,day,hour,minute,seconds);

Writeln ('Current date : ',Day:2,'/',Month:2,'/',Year:4);

Writeln ('Current time : ',Hour:2,':',minute:2,':',seconds:2);

end.

Execl

Synopsis: Execute process (using argument list)

Declaration: `procedure Execl(const Todo: String)
procedure Execl(const Todo: Ansistring)`

Visibility: default

Description: Replaces the currently running program with the program, specified in `path`. Path is split into a command and it's options. The executable in `path` is NOT searched in the path. The current environment is passed to the program. On success, `execl` does not return.

Errors: Errors are reported in `LinuxError`:

sys_eaccessFile is not a regular file, or has no execute permission. A component of the path has no search permission.

sys_epermThe file system is mounted *noexec*.

sys_e2bigArgument list too big.

sys_enoexecThe magic number in the file is incorrect.

sys_enoentThe file does not exist.

sys_enomemNot enough memory for kernel, or to split command line.

sys_enotdirA component of the path is not a directory.

sys_eloopThe path contains a circular reference (via symlinks).

See also: `Execve` (804), `Execv` (803), `Execvp` (805), `Execle` (801), `Execlp` (802), `Fork` (813)

Listing: `./olinuxex/ex10.pp`

Program `Example10`;

{ Program to demonstrate the Execl function. }

Uses `oldlinux` , `strings`;

begin

{ Execute 'ls -l', with current environment. }

{ 'ls' is NOT looked for in PATH environment variable. }

`Execl ('/bin/ls -l');`

end.

Execle

Synopsis: Execute process (using argument list, environment)

Declaration: `procedure Execle(Todo: String;Ep: ppchar)`
`procedure Execle(Todo: AnsiString;Ep: ppchar)`

Visibility: `default`

Description: Replaces the currently running program with the program, specified in `path`. Path is split into a command and it's options. The executable in `path` is searched in the path, if it isn't an absolute filename. The environment in `ep` is passed to the program. On success, `execle` does not return.

Errors: Errors are reported in `LinuxError`:

sys_eaccessFile is not a regular file, or has no execute permission. A component of the path has no search permission.

sys_epermThe file system is mounted *noexec*.

sys_e2bigArgument list too big.

sys_enoexecThe magic number in the file is incorrect.

sys_enoentThe file does not exist.

sys_enomemNot enough memory for kernel, or to split command line.

sys_enotdirA component of the path is not a directory.

sys_eloopThe path contains a circular reference (via symlinks).

See also: Execve (804), Execv (803), Execvp (805), Execl (800), Execlp (802), Fork (813)

Listing: ./olinuxex/ex11.pp

Program Example11;

{ Program to demonstrate the Execl function. }

Uses oldlinux , strings ;

begin

{ Execute 'ls -l', with current environment. }
{ 'ls' is NOT looked for in PATH environment variable. }
{ envp is defined in the system unit. }
 Execl ('/bin/ls -l', envp);

end.

Execlp

Synopsis: Execute process (using argument list, environment; search path)

Declaration: procedure Execlp(Todo: String;Ep: ppchar)
 procedure Execlp(Todo: Ansistring;Ep: ppchar)

Visibility: default

Description: Replaces the currently running program with the program, specified in path. Path is split into a command and it's options. The executable in path is searched in the path, if it isn't an absolute filename. The current environment is passed to the program. On success, execlp does not return.

Errors: Errors are reported in LinuxError:

sys_eaccessFile is not a regular file, or has no execute permission. A component of the path has no search permission.

sys_epermThe file system is mounted *noexec*.

sys_e2bigArgument list too big.

sys_enoexecThe magic number in the file is incorrect.

sys_enoentThe file does not exist.

sys_enomemNot enough memory for kernel, or to split command line.

sys_enotdirA component of the path is not a directory.

sys_eloopThe path contains a circular reference (via symlinks).

See also: Execve (804), Execv (803), Execvp (805), Execl (801), Execl (800), Fork (813)

Listing: ./olinuxex/ex12.pp

Program Example12;

{ Program to demonstrate the Execlp function. }

Uses oldlinux , strings ;

begin

{ Execute 'ls -l', with current environment. }
{ 'ls' is looked for in PATH environment variable. }
{ envp is defined in the system unit. }
 Execlp ('ls -l', envp);

end.

Execv

Synopsis: Execute process

Declaration: `procedure Execv(const path: PathStr; args: ppchar)`
`procedure Execv(const path: AnsiString; args: ppchar)`

Visibility: default

Description: Replaces the currently running program with the program, specified in path. It gives the program the options in args. This is a pointer to an array of pointers to null-terminated strings. The last pointer in this array should be nil. The current environment is passed to the program. On success, execv does not return.

Errors: Errors are reported in `LinuxError`:

sys_eaccessFile is not a regular file, or has no execute permission. A component of the path has no search permission.

sys_epermThe file system is mounted *noexec*.

sys_e2bigArgument list too big.

sys_enoexecThe magic number in the file is incorrect.

sys_enoentThe file does not exist.

sys_enomemNot enough memory for kernel.

sys_enotdirA component of the path is not a directory.

sys_eloopThe path contains a circular reference (via symlinks).

See also: `Execve` ([804](#)), `Execvp` ([805](#)), `Execl` ([801](#)), `Execl` ([800](#)), `Execlp` ([802](#)), `Fork` ([813](#))

Listing: ./olinuxex/ex8.pp

Program Example8;

{ Program to demonstrate the Execv function. }

Uses oldlinux , strings ;

Const Arg0 : PChar = '/bin/ls';
 Arg1 : Pchar = '-l';

Var PP : PPchar;

```

begin
  GetMem (PP,3*SizeOf(Pchar));
  PP[0]:=Arg0;
  PP[1]:=Arg1;
  PP[3]:=Nil;
  { Execute '/bin/ls -l', with current environment }
  Execv ('/bin/ls',pp);
end.

```

Execve

Synopsis: Execute process using environment

Declaration: `procedure Execve(Path: PathStr;args: ppchar;ep: ppchar)`
`procedure Execve(Path: AnsiString;args: ppchar;ep: ppchar)`
`procedure Execve(path: pchar;args: ppchar;ep: ppchar)`

Visibility: default

Description: Replaces the currently running program with the program, specified in `path`. It gives the program the options in `args`, and the environment in `ep`. They are pointers to an array of pointers to null-terminated strings. The last pointer in this array should be nil. On success, `execve` does not return.

Errors: Errors are reported in `LinuxError`:

sys_eaccessFile is not a regular file, or has no execute permission. A component of the path has no search permission.

sys_epermThe file system is mounted *noexec*.

sys_e2bigArgument list too big.

sys_enoexecThe magic number in the file is incorrect.

sys_enoentThe file does not exist.

sys_enomemNot enough memory for kernel.

sys_enotdirA component of the path is not a directory.

sys_eloopThe path contains a circular reference (via symlinks).

See also: `Execve` ([804](#)), `Execv` ([803](#)), `Execvp` ([805](#)), `Execle` ([801](#)), `Execl` ([800](#)), `Execlp` ([802](#)), `Fork` ([813](#))

Listing: `./olinuxex/ex7.pp`

Program Example7;

{ Program to demonstrate the Execve function. }

Uses oldlinux , strings;

Const Arg0 : PChar = '/bin/l~~s~~';
 Arg1 : PChar = '-l';

Var PP : PPchar;

begin

```

GetMem (PP,3*SizeOf(Pchar));
PP[0]:=Arg0;
PP[1]:=Arg1;
PP[3]:=Nil;
{ Execute '/bin/lis -l', with current environment }
{ Envp is defined in system.inc }
ExecVe ('/bin/lis',pp,envp);
end.

```

Execvp

Synopsis: Execute process, search path

Declaration: `procedure Execvp(Path: PathStr;Args: ppchar;Ep: ppchar)`
`procedure Execvp(Path: AnsiString;Args: ppchar;Ep: ppchar)`

Visibility: default

Description: Replaces the currently running program with the program, specified in path. The executable in path is searched in the path, if it isn't an absolute filename. It gives the program the options in args. This is a pointer to an array of pointers to null-terminated strings. The last pointer in this array should be nil. The current environment is passed to the program. On success, execvp does not return.

Errors: Errors are reported in `LinuxError`:

sys_eaccessFile is not a regular file, or has no execute permission. A component of the path has no search permission.

sys_epermThe file system is mounted *noexec*.

sys_e2bigArgument list too big.

sys_enoexecThe magic number in the file is incorrect.

sys_enoentThe file does not exist.

sys_enomemNot enough memory for kernel.

sys_enotdirA component of the path is not a directory.

sys_eloopThe path contains a circular reference (via symlinks).

See also: `Execve` ([804](#)), `Execv` ([803](#)), `Execle` ([801](#)), `Execl` ([800](#)), `Execlp` ([802](#)), `Fork` ([813](#))

Listing: `./olinuxex/ex9.pp`

Program Example9;

{ Program to demonstrate the Execvp function. }

Uses oldlinux , strings ;

Const Arg0 : PChar = 'ls' ;
Arg1 : PChar = '-l' ;

Var PP : PPchar ;

begin
GetMem (PP,3***SizeOf**(Pchar));

```

PP[0]:=Arg0;
PP[1]:=Arg1;
PP[3]:=Nil;
{ Execute 'ls -l', with current environment. }
{ 'ls' is looked for in PATH environment variable. }
{ Env is defined in the system unit. }
Execvp ('ls',pp,envp);
end.

```

ExitProcess

Synopsis: Exit the current process

Declaration: `procedure ExitProcess(val: LongInt)`

Visibility: default

Description: `ExitProcess` exits the currently running process, and report `Val` as the exit status.

Remark: If this call is executed, the normal unit finalization code will not be executed. This may lead to unexpected errors and stray files on your system. It is therefore recommended to use the `Halt` call instead.

Errors: None.

See also: `Fork` ([813](#)), `ExecVE` ([804](#))

Fcntl

Synopsis: File control operations.

Declaration: `function Fcntl(Fd: LongInt;Cmd: LongInt) : LongInt`
`procedure Fcntl(Fd: LongInt;Cmd: LongInt;Arg: LongInt)`
`function Fcntl(var Fd: Text;Cmd: LongInt) : LongInt`
`procedure Fcntl(var Fd: Text;Cmd: LongInt;Arg: LongInt)`

Visibility: default

Description: Read a file's attributes. `Fd` is an assigned file, or a valid file descriptor. `Cmd` specifies what to do, and is one of the following:

F_GetFdRead the `close_on_exec` flag. If the low-order bit is 0, then the file will remain open across `execve` calls.

F_GetFIRead the descriptor's flags.

F_GetOwnGet the Process ID of the owner of a socket.

F_SetFdSet the `close_on_exec` flag of `Fd`. (only the least significant bit is used).

F_GetLkReturn the `flock` record that prevents this process from obtaining the lock, or set the `l_type` field of the lock of there is no obstruction. `Arg` is a pointer to a `flock` record.

F_SetLkSet the lock or clear it (depending on `l_type` in the `flock` structure). if the lock is held by another process, an error occurs.

F_GetLkwSame as for **F_SetLk**, but wait until the lock is released.

F_SetOwnSet the Process or process group that owns a socket.

Errors: `LinuxError` is used to report errors.

sys_ebadf`Fd` has a bad file descriptor.

sys_eagain or **sys_eaccess**For **F_SetLk**, if the lock is held by another process.

fdClose

Synopsis: Close file descriptor

Declaration: `function fdClose(fd: LongInt) : Boolean`

Visibility: default

Description: `fdClose` closes a file with file descriptor `Fd`. The function returns `True` if the file was closed successfully, `False` otherwise.

For an example, see `fdOpen` (807).

Errors: Errors are returned in `LinuxError`.

See also: `fdOpen` (807), `fdRead` (808), `fdWrite` (810), `fdTruncate` (810), `fdFlush` (807), `fdSeek` (809)

fdFlush

Synopsis: Flush kernel file buffer

Declaration: `function fdFlush(fd: LongInt) : Boolean`

Visibility: default

Description: `fdflush` flushes the Linux kernel file buffer, so the file is actually written to disk. This is NOT the same as the internal buffer, maintained by Free Pascal. The function returns `True` if the call was successful, `false` if an error occurred.

For an example, see `fdRead` (808).

Errors: Errors are returned in `LinuxError`.

See also: `fdOpen` (807), `fdClose` (807), `fdRead` (808), `fdWrite` (810), `fdTruncate` (810), `fdSeek` (809)

fdOpen

Synopsis: Open file and return file descriptor

Declaration: `function fdOpen(pathname: String; flags: LongInt) : LongInt`
`function fdOpen(pathname: String; flags: LongInt; mode: LongInt) : LongInt`
`function fdOpen(pathname: pchar; flags: LongInt) : LongInt`
`function fdOpen(pathname: pchar; flags: LongInt; mode: LongInt) : LongInt`

Visibility: default

Description: `fdOpen` opens a file in `PathName` with flags `flags` One of the following:

Open_RdOnlyFile is opened Read-only

Open_WrOnlyFile is opened Write-only

Open_RdWrFile is opened Read-Write

The flags may be OR-ed with one of the following constants:

Open_CreatFile is created if it doesn't exist.

Open_ExclIf the file is opened with `Open_Creat` and it already exists, the call will fail.

Open_NoCttyIf the file is a terminal device, it will NOT become the process' controlling terminal.

Open_TruncIf the file exists, it will be truncated.

Open_Appendthe file is opened in append mode. *Before each write*, the file pointer is positioned at the end of the file.

Open_NonBlockThe file is opened in non-blocking mode. No operation on the file descriptor will cause the calling process to wait till.

Open_NDelayIdem as **Open_NonBlock**

Open_SyncThe file is opened for synchronous IO. Any write operation on the file will not return untill the data is physically written to disk.

Open_NoFollowif the file is a symbolic link, the open fails. (linux 2.1.126 and higher only)

Open_Directoryif the file is not a directory, the open fails. (linux 2.1.126 and higher only)

PathName can be of type PChar or String. The optional mode argument specifies the permissions to set when opening the file. This is modified by the umask setting. The real permissions are Mode and not umask. The return value of the function is the filedescriptor, or a negative value if there was an error.

Errors: Errors are returned in LinuxError.

See also: fdClose (807), fdRead (808), fdWrite (810), fdTruncate (810), fdFlush (807), fdSeek (809)

Listing: ./olinuxex/ex19.pp

Program Example19;

{ Program to demonstrate the fdOpen, fdwrite and fdClose functions. }

Uses oldlinux;

Const Line : **String**[80] = 'This is easy writing !';

Var FD : Longint;

begin

 FD:=fdOpen ('Test.dat',Open_WrOnly or Open_Creat);

if FD>0 **then**

begin

if length(Line)<>fdwrite (FD,Line[1],Length(Line)) **then**

 WriteLn ('Error when writing to file !');

 fdClose(FD);

end;

end.

fdRead

Synopsis: Read data from file descriptor

Declaration: function fdRead(fd: LongInt;var buf;size: LongInt) : LongInt

Visibility: default

Description: fdRead reads at most size bytes from the file descriptor fd, and stores them in buf. The function returns the number of bytes actually read, or -1 if an error occurred. No checking on the length of buf is done.

Errors: Errors are returned in LinuxError.

See also: fdOpen (807), fdClose (807), fdWrite (810), fdTruncate (810), fdFlush (807), fdSeek (809)

Listing: ./olinuxex/ex20.pp

Program Example20;

{ Program to demonstrate the fdRead and fdTruncate functions. }

Uses oldlinux;

Const Data : **string**[10] = '12345687890';

Var FD : Longint;
 I : longint;

```
begin
  FD:=fdOpen('test.dat',open_wronly or open_creat,octal(666));
  if fd>0 then
    begin
      { Fill file with data }
      for I:=1 to 10 do
        if fdWrite (FD,Data[I],10)<>10 then
          begin
            writeln ('Error when writing !');
            halt(1);
          end;
      fdClose(FD);
      FD:=fdOpen('test.dat',open_rdonly);
      { Read data again }
      If FD>0 then
        begin
          For I:=1 to 5 do
            if fdRead (FD,Data[I],10)<>10 then
              begin
                Writeln ('Error when Reading !');
                Halt(2);
              end;
          fdClose(FD);
          { Truncating file at 60 bytes }
          { For truncating , file must be open or write }
          FD:=fdOpen('test.dat',open_wronly,octal(666));
          if FD>0 then
            begin
              if not fdTruncate(FD,60) then
                Writeln('Error when truncating !');
              fdClose (FD);
            end;
          end;
        end;
      end;
    end.
```

fdSeek

Synopsis: Set file pointer position.

Declaration: function fdSeek(fd: LongInt;pos: LongInt;seektype: LongInt) : LongInt

Visibility: default

Description: `fdSeek` sets the current fileposition of file `fd` to `Pos`, starting from `SeekType`, which can be one of the following:

`Seek_SetPos` is the absolute position in the file.

`Seek_CurPos` is relative to the current position.

`Seek_endPos` is relative to the end of the file.

The function returns the new fileposition, or -1 of an error occurred.

For an example, see `fdOpen` (807).

Errors: Errors are returned in `LinuxError`.

See also: `fdOpen` (807), `fdWrite` (810), `fdClose` (807), `fdRead` (808), `fdTruncate` (810), `fdFlush` (807)

fdTruncate

Synopsis: Truncate file on certain size.

Declaration: `function fdTruncate(fd: LongInt;size: LongInt) : Boolean`

Visibility: default

Description: `fdTruncate` sets the length of a file in `fd` on `size` bytes, where `size` must be less than or equal to the current length of the file in `fd`. The function returns `True` if the call was successful, `false` if an error occurred.

Errors: Errors are returned in `LinuxError`.

See also: `fdOpen` (807), `fdClose` (807), `fdRead` (808), `fdWrite` (810), `fdFlush` (807), `fdSeek` (809)

fdWrite

Synopsis: Write data to file descriptor

Declaration: `function fdWrite(fd: LongInt;const buf;size: LongInt) : LongInt`

Visibility: default

Description: `fdWrite` writes at most `size` bytes from `buf` to file descriptor `fd`. The function returns the number of bytes actually written, or -1 if an error occurred.

Errors: Errors are returned in `LinuxError`.

See also: `fdOpen` (807), `fdClose` (807), `fdRead` (808), `fdTruncate` (810), `fdSeek` (809), `fdFlush` (807)

FD_Clr

Synopsis: Clears a filedescriptor in a set

Declaration: `procedure FD_Clr(fd: LongInt;var fds: fdSet)`

Visibility: default

Description: `FD_Clr` clears file descriptor `fd` in filedescriptor set `fds`.

For an example, see `Select` (841).

Errors: None.

See also: `Select` (841), `SelectText` (842), `GetFS` (821), `FD_ZERO` (811), `FD_Set` (811), `FD_IsSet` (811)

FD_IsSet

Synopsis: Check whether a filedescriptor is set

Declaration: `function FD_IsSet(fd: LongInt;var fds: fdSet) : Boolean`

Visibility: default

Description: `FD_Set` Checks whether file descriptor `fd` in filedescriptor set `fds` is set.

For an example, see [Select \(841\)](#).

Errors: None.

See also: [Select \(841\)](#), [SelectText \(842\)](#), [GetFS \(821\)](#), [FD_ZERO \(811\)](#), [FD_Clr \(810\)](#), [FD_Set \(811\)](#)

FD_Set

Synopsis: Set a filedescriptor in a set

Declaration: `procedure FD_Set(fd: LongInt;var fds: fdSet)`

Visibility: default

Description: `FD_Set` sets file descriptor `fd` in filedescriptor set `fds`.

For an example, see [Select \(841\)](#).

Errors: None.

See also: [Select \(841\)](#), [SelectText \(842\)](#), [GetFS \(821\)](#), [FD_ZERO \(811\)](#), [FD_Clr \(810\)](#), [FD_IsSet \(811\)](#)

FD_Zero

Synopsis: Clear all file descriptors in set

Declaration: `procedure FD_Zero(var fds: fdSet)`

Visibility: default

Description: `FD_ZERO` clears all the filedescriptors in the file descriptor set `fds`.

For an example, see [Select \(841\)](#).

Errors: None.

See also: [Select \(841\)](#), [SelectText \(842\)](#), [GetFS \(821\)](#), [FD_Clr \(810\)](#), [FD_Set \(811\)](#), [FD_IsSet \(811\)](#)

FExpand

Synopsis: Expand filename to fully qualified path

Declaration: `function FExpand(const Path: PathStr) : PathStr`

Visibility: default

Description: `FExpand` expands `Path` to a full path, starting from root, eliminating directory references such as `.` and `..` from the result.

Errors: None

See also: [BaseName \(791\)](#), [DirName \(798\)](#)

Listing: ./olinuxex/ex45.pp

Program Example45;

```
{ Program to demonstrate the FExpand function. }
```

Uses oldlinux;

begin

```
  WriteLn ( 'This program is in : ',FExpand(Paramstr(0)));
end.
```

Flock

Synopsis: Lock a file (advisory lock)

Declaration: function Flock(fd: LongInt;mode: LongInt) : Boolean
 function Flock(var T: text;mode: LongInt) : Boolean
 function Flock(var F: File;mode: LongInt) : Boolean

Visibility: default

Description: FLock implements file locking. it sets or removes a lock on the file F. F can be of type Text or File, or it can be a linux filedescriptor (a longint) Mode can be one of the following constants :

LOCK_SHsets a shared lock.

LOCK_EXsets an exclusive lock.

LOCK_UNunlocks the file.

LOCK_NBThis can be OR-ed together with the other. If this is done the application doesn't block when locking.

The function returns True if successful, False otherwise.

Errors: If an error occurs, it is reported in LinuxError.

See also: Fcntl ([806](#))

FNMatch

Synopsis: Check whether filename matches wildcard specification

Declaration: function FNMatch(const Pattern: String;const Name: String) : Boolean

Visibility: default

Description: FNMatch returns True if the filename in Name matches the wildcard pattern in Pattern, False otherwise.

Pattern can contain the wildcards * (match zero or more arbitrary characters) or ? (match a single character).

Errors: None.

See also: FSearch ([814](#)), FExpand ([811](#))

Listing: ./olinuxex/ex69.pp

```

Program Example69;

{ Program to demonstrate the FNMatch function. }

Uses oldlinux;

Procedure TestMatch(Pattern,Name : String);

begin
  Write ( ' ',Name, ' ');
  If FNMatch ( Pattern,Name) then
    Write ( 'matches')
  else
    Write ( 'does not match');
  WriteLn( ' ',Pattern, ' '. );
end;

begin
  TestMatch( '*', 'FileName' );
  TestMatch( '.*', 'FileName' );
  TestMatch( '*a*', 'FileName' );
  TestMatch( '?ile*', 'FileName' );
  TestMatch( '?', 'FileName' );
  TestMatch( '.?', 'FileName' );
  TestMatch( '?a*', 'FileName' );
  TestMatch( '??*me?', 'FileName' );
end.

```

Fork

Synopsis: Create child process

Declaration: `function Fork : LongInt`

Visibility: default

Description: `Fork` creates a child process which is a copy of the parent process. `Fork` returns the process ID in the parent process, and zero in the child's process. (you can get the parent's PID with `GetPPid` ([823](#))).

Errors: On error, -1 is returned to the parent, and no child is created.

`sys_eagain` Not enough memory to create child process.

See also: `Execve` ([804](#)), `Clone` ([795](#))

FReName

Synopsis: Rename file

Declaration: `function FReName(OldName: Pchar;NewName: Pchar) : Boolean`
`function FReName(OldName: String;NewName: String) : Boolean`

Visibility: default

Description: `FRename` renames the file `OldName` to `NewName`. `NewName` can be in a different directory than `OldName`, but it cannot be on another partition (device). Any existing file on the new location will be replaced.

If the operation fails, then the `OldName` file will be preserved.

The function returns `True` on succes, `False` on failure.

Errors: On error, errors are reported in `LinuxError`. Possible errors include:

`sys_eisdir``NewName` exists and is a directory, but `OldName` is not a directory.

`sys_exdev``NewName` and `OldName` are on different devices.

`sys_enotempty` or `sys_eexist``NewName` is an existing, non-empty directory.

`sys_ebusy``OldName` or `NewName` is a directory and is in use by another process.

`sys_einval``NewName` is part of `OldName`.

`sys_emlink``OldPath` or `NewPath` already have the maximum amount of links pointing to them.

`sys_enotdir`part of `OldName` or `NewName` is not directory.

`sys_efault`For the `pchar` case: One of the pointers points to an invalid address.

`sys_eaccess`access is denied when attempting to move the file.

`sys_enametoolong`Either `OldName` or `NewName` is too long.

`sys_enoent`a directory component in `OldName` or `NewName` didn't exist.

`sys_enomem`not enough kernel memory.

`sys_erofs``NewName` or `OldName` is on a read-only file system.

`sys_eloop`too many symbolic links were encountered trying to expand `OldName` or `NewName`

`sys_enospc`the filesystem has no room for the new directory entry.

See also: `UnLink` ([859](#))

FSearch

Synopsis: Search for file in search path.

Declaration: `function FSearch(const path: PathStr;dirlist: String) : PathStr`

Visibility: default

Description: `FSearch` searches in `DirList`, a colon separated list of directories, for a file named `Path`. It then returns a path to the found file.

Errors: An empty string if no such file was found.

See also: `BaseName` ([791](#)), `DirName` ([798](#)), `FExpand` ([811](#)), `FNMatch` ([812](#))

Listing: `./olinuxex/ex46.pp`

Program `Example46`;

{ Program to demonstrate the FSearch function. }

Uses `oldlinux` , `strings` ;

begin

WriteLn ('Is is in : ',FSearch ('Is' ,**strpas**(Getenv('PATH'))));

end.

FSplit

Synopsis: Split filename into path, name and extension

Declaration: `procedure FSplit(const Path: PathStr; var Dir: DirStr; var Name: NameStr;
var Ext: ExtStr)`

Visibility: default

Description: `FSplit` splits a full file name into 3 parts : A `Path`, a `Name` and an extension (in `ext`). The extension is taken to be all letters after the last dot (.).

Errors: None.

See also: `FSearch` ([814](#))

Listing: `./olinuxex/ex67.pp`

Program `Example67;`

`uses oldlinux;`

`{ Program to demonstrate the FSplit function. }`

`var`

`Path, Name, Ext : string;`

`begin`

`FSplit(ParamStr(1), Path, Name, Ext);`

`WriteLn('Split ', ParamStr(1), ' in:');`

`WriteLn('Path : ', Path);`

`WriteLn('Name : ', Name);`

`WriteLn('Extension: ', Ext);`

`end.`

FSStat

Synopsis: Retrieve filesystem information.

Declaration: `function FSStat(Path: PathStr; var Info: Statfs) : Boolean`
`function FSStat(Fd: LongInt; var Info: Statfs) : Boolean`

Visibility: default

Description: `FSStat` returns in `Info` information about the filesystem on which the file `Path` resides, or on which the file with file descriptor `fd` resides. `Info` is of type `statfs`. The function returns `True` if the call was successful, `False` if the call failed.

Errors: `LinuxError` is used to report errors.

sys_enotdir A component of `Path` is not a directory.

sys_einval Invalid character in `Path`.

sys_enoent `Path` does not exist.

sys_eaccess Search permission is denied for component in `Path`.

sys_eloop A circular symbolic link was encountered in `Path`.

sys_eio An error occurred while reading from the filesystem.

See also: FStat (816), LStat (831)

Listing: ./olinuxex/ex30.pp

```

program Example30;

{ Program to demonstrate the FSStat function. }

uses oldlinux;

var s : string;
    info : statfs;

begin
  writeln ('Info about current partition : ');
  s := '.';
  while s <> 'q' do
    begin
      if not fsstat (s,info) then
        begin
          writeln ('Fstat failed. Errno : ',linuxerror);
          halt (1);
        end;
      writeln;
      writeln ('Result of fsstat on file ',s,'. ');
      writeln ('fstype   : ',info.fstype);
      writeln ('bsize    : ',info.bsize);
      writeln ('bfree    : ',info.bfree);
      writeln ('bavail   : ',info.bavail);
      writeln ('files    : ',info.files);
      writeln ('ffree    : ',info.ffree);
      writeln ('fsid     : ',info.fsid);
      writeln ('Namelen  : ',info.namelen);
      write ('Type name of file to do fsstat. (q quits) : ');
      readln (s)
    end;
  end.

```

FStat

Synopsis: Retrieve information about a file

Declaration: function FStat(Path: PathStr;var Info: Stat) : Boolean
 function FStat(Fd: LongInt;var Info: Stat) : Boolean
 function FStat(var F: Text;var Info: Stat) : Boolean
 function FStat(var F: File;var Info: Stat) : Boolean

Visibility: default

Description: FStat gets information about the file specified in one of the following:

Patha file on the filesystem.

Fda valid file descriptor.

Fan opened text file or untyped file.

and stores it in Info, which is of type stat. The function returns True if the call was successful, False if the call failed.

Errors: `LinuxError` is used to report errors.

`sys_enoentPath` does not exist.

See also: `FStat` (815), `LStat` (831)

Listing: `./olinuxex/ex28.pp`

```

program example28;

{ Program to demonstrate the FStat function. }

uses oldlinux;

var f : text;
    i : byte;
    info : stat;

begin
  { Make a file }
  assign (f, 'test.fil ');
  rewrite (f);
  for i:=1 to 10 do writeln (f, 'Testline # ', i);
  close (f);
  { Do the call on made file. }
  if not fstat ('test.fil ', info) then
    begin
      writeln ('Fstat failed. Errno : ', linuxerror);
      halt (1);
    end;
  writeln;
  writeln ('Result of fstat on file ''test.fil ''.');
  writeln ('Inode    : ', info.ino);
  writeln ('Mode     : ', info.mode);
  writeln ('nlink    : ', info.nlink);
  writeln ('uid      : ', info.uid);
  writeln ('gid      : ', info.gid);
  writeln ('rdev     : ', info.rdev);
  writeln ('Size     : ', info.size);
  writeln ('Blksize  : ', info.blksize);
  writeln ('Blocks   : ', info.blocks);
  writeln ('atime    : ', info.atime);
  writeln ('mtime    : ', info.mtime);
  writeln ('ctime    : ', info.ctime);
  { Remove file }
  erase (f);
end.

```

GetDate

Synopsis: Return the system date

Declaration: `procedure GetDate(var Year: Word; var Month: Word; var Day: Word)`

Visibility: default

Description: Returns the current date.

Errors: None

See also: [GetEpochTime \(820\)](#), [GetTime \(824\)](#), [GetDateTime \(818\)](#), [EpochToLocal \(800\)](#)

Listing: ./olinuxex/ex6.pp

Program Example6;

{ Program to demonstrate the GetDate function. }

Uses oldlinux;

Var Year, Month, Day : Word;

begin

GetDate (Year, Month, Day);

WriteIn ('Date : ', Day:2, ' / ', Month:2, ' / ', Year:4);

end.

GetDateTime

Synopsis: Return system date and time

Declaration: `procedure GetDateTime(var Year: Word; var Month: Word; var Day: Word;
var hour: Word; var minute: Word; var second: Word)`

Visibility: default

Description: Returns the current date and time. The time is corrected for the local time zone. This procedure is equivalent to the [GetDate \(817\)](#) and [GetTime](#) calls.

Errors: None

See also: [GetEpochTime \(820\)](#), [GetTime \(824\)](#), [EpochToLocal \(800\)](#), [GetDate \(817\)](#)

Listing: ./olinuxex/ex60.pp

Program Example6;

{ Program to demonstrate the GetDateTime function. }

Uses oldlinux;

Var Year, Month, Day, Hour, min, sec : Word;

begin

GetDateTime (Year, Month, Day, Hour, min, sec);

WriteIn ('Date : ', Day:2, ' / ', Month:2, ' / ', Year:4);

WriteIn ('Time : ', Hour:2, ': ', Min:2, ': ', Sec:2);

end.

GetDomainName

Synopsis: Return current domain name

Declaration: `function GetDomainName : String`

Visibility: default

Description: Get the domain name of the machine on which the process is running. An empty string is returned if the domain is not set.

Errors: None.

See also: `GetHostName` ([822](#))

Listing: ./olinuxex/ex39.pp

Program Example39;

{ Program to demonstrate the GetDomainName function. }

Uses oldlinux;

begin

WriteLn ('Domain name of this machine is : ', GetDomainName);
end.

GetEGid

Synopsis: Return effective group ID

Declaration: `function GetEGid : LongInt`

Visibility: default

Description: Get the effective group ID of the currently running process.

Errors: None.

See also: `GetGid` ([821](#))

Listing: ./olinuxex/ex18.pp

Program Example18;

{ Program to demonstrate the GetGid and GetEGid functions. }

Uses oldlinux;

begin

writeln ('Group Id = ', getgid, ' Effective group Id = ', getegid);
end.

GetEnv

Synopsis: Return value of environment variable.

Declaration: `function GetEnv(P: String) : Pchar`

Visibility: default

Description: `GetEnv` returns the value of the environment variable in P. If the variable is not defined, nil is returned. The value of the environment variable may be the empty string. A PChar is returned to accomodate for strings longer than 255 bytes, `TERMCAP` and `LS_COLORS`, for instance.

Errors: None.

Listing: ./olinuxex/ex41.pp

```
Program Example41;

{ Program to demonstrate the GetEnv function. }

Uses oldlinux;

begin
  WriteLn ( 'Path is : ',Getenv( 'PATH' ));
end.
```

GetEpochTime

Synopsis: Return the current unix time

Declaration: function GetEpochTime : LongInt

Visibility: default

Description: returns the number of seconds since 00:00:00 gmt, january 1, 1970. it is adjusted to the local time zone, but not to DST.

Errors: no errors

See also: EpochToLocal ([800](#)), GetTime ([824](#))

Listing: ./olinuxex/ex1.pp

```
Program Example1;

{ Program to demonstrate the GetEpochTime function. }

Uses oldlinux;

begin
  Write ( 'Secs past the start of the Epoch (00:00 1/1/1980) : ');
  WriteLn ( GetEpochTime );
end.
```

GetEUid

Synopsis: Return effective user ID

Declaration: function GetEUid : LongInt

Visibility: default

Description: Get the effective user ID of the currently running process.

Errors: None.

See also: GetUid ([825](#))

Listing: ./olinuxex/ex17.pp

Program Example17;

{ Program to demonstrate the GetUid and GetEUid functions. }

Uses oldlinux;

begin

writeln ('User Id = ',getuid, ' Effective user Id = ',geteuid);
end.

GetFS

Synopsis: Return file selector

Declaration: function GetFS(var T: Text) : LongInt
 function GetFS(var F: File) : LongInt

Visibility: default

Description: GetFS returns the file selector that the kernel provided for your file. In principle you don't need this file selector. Only for some calls it is needed, such as the Select (841) call or so.

Errors: In case the file was not opened, then -1 is returned.

See also: Select (841)

Listing: ./olinuxex/ex34.pp

Program Example33;

{ Program to demonstrate the SelectText function. }

Uses oldlinux;

Var tv : TimeVal;

begin

Writeln ('Press the <ENTER> to continue the program. ');
 { Wait until File descriptor 0 (=Input) changes }
 SelectText (Input, nil);
 { Get rid of <ENTER> in buffer }
 readln ;
 Writeln ('Press <ENTER> key in less than 2 seconds... ');
 tv.sec:=2;
 tv.usec:=0;
 if SelectText (Input, @tv) > 0 **then**
 Writeln ('Thank you ! ')
 else
 Writeln ('Too late ! ');
end.

GetGid

Synopsis: Return real group ID

Declaration: function GetGid : LongInt

Visibility: default

Description: Get the real group ID of the currently running process.

Errors: None.

See also: GetEGid ([819](#))

Listing: ./olinuxex/ex18.pp

Program Example18;

{ Program to demonstrate the GetGid and GetEGid functions. }

Uses oldlinux;

```
begin
  writeln ( 'Group Id = ',getgid,' Effective group Id = ',getegid);
end.
```

GetHostName

Synopsis: Return host name

Declaration: function GetHostName : String

Visibility: default

Description: Get the hostname of the machine on which the process is running. An empty string is returned if hostname is not set.

Errors: None.

See also: GetDomainName ([818](#))

Listing: ./olinuxex/ex40.pp

Program Example40;

{ Program to demonstrate the GetHostName function. }

Uses oldlinux;

```
begin
  WriteLn ( 'Name of this machine is : ',GetHostName);
end.
```

GetLocalTimezone

Synopsis: Return local timzeone information

Declaration: procedure GetLocalTimezone(timer: LongInt;var leap_correct: LongInt;
var leap_hit: LongInt)
procedure GetLocalTimezone(timer: LongInt)

Visibility: default

Description: `GetLocalTimeZone` returns the local timezone information. It also initializes the `TZSeconds` variable, which is used to correct the epoch time to local time.

There should never be any need to call this function directly. It is called by the initialization routines of the Linux unit.

See also: `GetTimezoneFile` (825), `ReadTimezoneFile` (840)

GetPid

Synopsis: Return current process ID

Declaration: `function GetPid : LongInt`

Visibility: default

Description: Get the Process ID of the currently running process.

Errors: None.

See also: `GetPPid` (823)

Listing: `./olinuxex/ex16.pp`

Program `Example16;`

{ Program to demonstrate the GetPid , GetPPid function . }

Uses `oldlinux;`

begin

WriteLn ('Process Id = ',getpid , ' Parent process Id = ',getppid);

end.

GetPPid

Synopsis: Return parent process ID

Declaration: `function GetPPid : LongInt`

Visibility: default

Description: Get the Process ID of the parent process.

Errors: None.

See also: `GetPid` (823)

Listing: `./olinuxex/ex16.pp`

Program `Example16;`

{ Program to demonstrate the GetPid , GetPPid function . }

Uses `oldlinux;`

begin

WriteLn ('Process Id = ',getpid , ' Parent process Id = ',getppid);

end.

GetPriority

Synopsis: Return process priority

Declaration: `function GetPriority(Which: Integer;Who: Integer) : Integer`

Visibility: default

Description: GetPriority returns the priority with which a process is running. Which process(es) is determined by the Which and Who variables. Which can be one of the pre-defined Prio_Process, Prio_PGrp, Prio_User, in which case Who is the process ID, Process group ID or User ID, respectively.

For an example, see Nice (835).

Errors: Error checking must be done on LinuxError, since a priority can be negative.

sys_esrchNo process found using which and who.

sys_einvalWhich was not one of Prio_Process, Prio_Grp or Prio_User.

See also: SetPriority (843), Nice (835)

GetTime

Synopsis: Return current system time

Declaration: `procedure GetTime(var hour: Word;var min: Word;var sec: Word;
var msec: Word;var usec: Word)
procedure GetTime(var hour: Word;var min: Word;var sec: Word;
var sec100: Word)
procedure GetTime(var hour: Word;var min: Word;var sec: Word)`

Visibility: default

Description: Returns the current time of the day, adjusted to local time. Upon return, the parameters are filled with

hourHours since 00:00 today.

minminutes in current hour.

secseconds in current minute.

sec100hundreds of seconds in current second.

msecmilliseconds in current second.

usecmicroseconds in current second.

Errors: None

See also: GetEpochTime (820), GetDate (817), GetDateTime (818), EpochToLocal (800)

Listing: ./olinuxex/ex5.pp

Program Example5;

{ Program to demonstrate the GetTime function. }

Uses oldlinux;

Var Hour, Minute, Second : Word;

```

begin
  GetTime (Hour, Minute, Second);
  WriteLn ('Time : ', Hour:2, ': ', Minute:2, ': ', Second:2);
end.

```

GetTimeOfDay

Synopsis: Return kernel time of day in GMT

Declaration: `procedure GetTimeOfDay(var tv: timeval)`
`function GetTimeOfDay : LongInt`

Visibility: default

Description: `GetTimeOfDay` returns the number of seconds since 00:00, January 1 1970, GMT in a `timeval` record. This time NOT corrected any way, not taking into account timezones, daylight savings time and so on.

It is simply a wrapper to the kernel system call. To get the local time, `GetTime` (824).

Errors: None.

See also: `GetTime` (824), `GetTimeOfDay` (825)

GetTimezoneFile

Synopsis: Return name of timezone information file

Declaration: `function GetTimezoneFile : String`

Visibility: default

Description: `GetTimezoneFile` returns the location of the current timezone file. The location of file is determined as follows:

- 1.If `/etc/timezone` exists, it is read, and the contents of this file is returned. This should work on Debian systems.
- 2.If `/usr/lib/zoneinfo/localtime` exists, then it is returned. (this file is a symlink to the timezone file on SuSE systems)
- 3.If `/etc/localtime` exists, then it is returned. (this file is a symlink to the timezone file on RedHat systems)

Errors: If no file was found, an empty string is returned.

See also: `ReadTimezoneFile` (840)

GetUid

Synopsis: Return current user ID

Declaration: `function GetUid : LongInt`

Visibility: default

Description: Get the real user ID of the currently running process.

Errors: None.

See also: [GetEUid \(820\)](#)

Listing: ./olinuxex/ex17.pp

Program Example17;

{ Program to demonstrate the GetUid and GetEUid functions. }

Uses oldlinux;

begin
 writeln ('User Id = ',getuid,' Effective user Id = ',geteuid);
end.

Glob

Synopsis: Find filenames matching a wildcard pattern

Declaration: `function Glob(const path: PathStr) : pglob`

Visibility: default

Description: Glob returns a pointer to a glob structure which contains all filenames which exist and match the pattern in Path. The pattern can contain wildcard characters, which have their usual meaning.

Errors: Returns nil on error, and `LinuxError` is set.

sys_enomemNo memory on heap for glob structure.

othersAs returned by the `opendir` call, and `sys_readdir`.

See also: [GlobFree \(827\)](#)

Listing: ./olinuxex/ex49.pp

Program Example49;

{ Program to demonstrate the Glob and GlobFree functions. }

Uses oldlinux;

Var G1,G2 : PGlob;

begin
 G1:=Glob ('*');
 if LinuxError=0 **then**
 begin
 G2:=G1;
 writeln ('Files in this directory : ');
 While g2<>Nil **do**
 begin
 writeln (g2^.name);
 g2:=g2^.next;
 end;
 GlobFree (g1);
 end;
end.

Globfree

Synopsis: Free result of Glob (826) call

Declaration: `procedure Globfree(var p: pglob)`

Visibility: default

Description: Releases the memory, occupied by a pglob structure. P is set to nil.

For an example, see Glob (826).

Errors: None

See also: Glob (826)

IOctl

Synopsis: General kernel IOCTL call.

Declaration: `function IOctl(Handle: LongInt;Ndx: LongInt;Data: Pointer) : Boolean`

Visibility: default

Description: This is a general interface to the Unix/ linux ioctl call. It performs various operations on the filedescriptor Handle. Ndx describes the operation to perform. Data points to data needed for the Ndx function. The structure of this data is function-dependent, so we don't elaborate on this here. For more information on this, see various manual pages under linux.

Errors: Errors are reported in LinuxError. They are very dependent on the used function, that's why we don't list them here

Listing: ./olinuxex/ex54.pp

Program Example54;

uses oldlinux;

{ Program to demonstrate the IOctl function . }

var

tios : Termios;

begin

IOctl(1,TCGETS,@tios);

WriteLn('Input Flags : \$',hexstr(tios.c_iflag,8));

WriteLn('Output Flags : \$',hexstr(tios.c_oflag,8));

WriteLn('Line Flags : \$',hexstr(tios.c_lflag,8));

WriteLn('Control Flags: \$',hexstr(tios.c_cflag,8));

end.

IOperm

Synopsis: Set permission on IO ports

Declaration: `function IOperm(From: Cardinal;Num: Cardinal;Value: LongInt) : Boolean`

Visibility: default

Description: `IOperm` sets permissions on `Num` ports starting with port `From` to `Value`. The function returns `True` if the call was successful, `False` otherwise.

Note:

- This works ONLY as root.
- Only the first `0x03ff` ports can be set.
- When doing a `Fork` (813), the permissions are reset. When doing a `Execve` (804) they are kept.

Errors: Errors are returned in `LinuxError`

IoPL

Synopsis: Set I/O privilege level

Declaration: `function IoPL(Level: LongInt) : Boolean`

Visibility: default

Description: `IoPL` sets the I/O privilege level. It is intended for completeness only, one should normally not use it.

IsATTY

Synopsis: Check if filehandle is a TTY (terminal)

Declaration: `function IsATTY(Handle: LongInt) : Boolean`
`function IsATTY(var f: text) : Boolean`

Visibility: default

Description: Check if the filehandle described by `f` is a terminal. `f` can be of type

1. `longint` for file handles;
2. `Text` for text variables such as input etc.

Returns `True` if `f` is a terminal, `False` otherwise.

Errors: No errors are reported

See also: `IOctl` (827), `TTYName` (858)

Kill

Synopsis: Send a signal to a process

Declaration: `function Kill(Pid: LongInt; Sig: LongInt) : Integer`

Visibility: default

Description: Send a signal `Sig` to a process or process group. If `Pid > 0` then the signal is sent to `Pid`, if it equals `-1`, then the signal is sent to all processes except process 1. If `Pid < -1` then the signal is sent to process group `-Pid`. The return value is zero, except in case three, where the return value is the number of processes to which the signal was sent.

Errors: `LinuxError` is used to report errors:

sys_einvalAn invalid signal is sent.

sys_esrchThe Pid or process group don't exist.

sys_epermThe effective userid of the current process doesn't math the one of process Pid.

See also: SigAction (844), Signal (846)

Link

Synopsis: Create a hard link to a file

Declaration: `function Link(OldPath: PathStr;NewPath: PathStr) : Boolean`

Visibility: default

Description: Link makes NewPath point to the same file als OldPath. The two files then have the same inode number. This is known as a 'hard' link. The function returns True if the call was succesfull, False if the call failed.

Errors: Errors are returned in `LinuxError`.

sys_exdevOldPath and NewPath are not on the same filesystem.

sys_epermThe filesystem containing oldpath and newpath doesn't support linking files.

sys_eaccessWrite access for the directory containing Newpath is disallowed, or one of the directories in OldPath or {NewPath} has no search (=execute) permission.

sys_enoentA directory entry in OldPath or NewPath does not exist or is a symbolic link pointing to a non-existent directory.

sys_enotdirA directory entry in OldPath or NewPath is nor a directory.

sys_enomemInsufficient kernel memory.

sys_erofsThe files are on a read-only filesystem.

sys_eexistNewPath already exists.

sys_emlinkOldPath has reached maximal link count.

sys_eloopOldPath or NewPath has a reference to a circular symbolic link, i.e. a symbolic link, whose expansion points to itself.

sys_enospcThe device containing NewPath has no room for another entry.

sys_epermOldPath points to . or .. of a directory.

See also: SymLink (849), UnLink (859)

Listing: ./olinuxex/ex21.pp

Program Example21;

{ Program to demonstrate the Link and UnLink functions. }

Uses oldlinux;

Var F : Text;

 S : **String**;

begin

 Assign (F, 'test.txt');

 Rewrite (F);

 WriteLn (F, 'This is written to test.txt');

```

Close(f);
{ new.txt and test.txt are now the same file }
if not Link ('test.txt', 'new.txt') then
  writeln ('Error when linking !');
{ Removing test.txt still leaves new.txt }
If not Unlink ('test.txt') then
  Writeln ('Error when unlinking !');
Assign (f, 'new.txt ');
Reset (F);
While not EOF(f) do
  begin
    Readln(F,S);
    Writeln ('> ',s);
  end;
Close (f);
{ Remove new.txt also }
If not Unlink ('new.txt') then
  Writeln ('Error when unlinking !');
end.

```

LocalToEpoch

Synopsis: Convert local time to epoch (unix) time

Declaration: `function LocalToEpoch(year: Word;month: Word;day: Word;hour: Word;
minute: Word;second: Word) : LongInt`

Visibility: default

Description: Converts the Local time to epoch time (=Number of seconds since 00:00:00 , January 1, 1970).

Errors: None

See also: [GetEpochTime \(820\)](#), [EpochToLocal \(800\)](#), [GetTime \(824\)](#), [GetDate \(817\)](#)

Listing: ./olinuxex/ex4.pp

Program Example4;

{ Program to demonstrate the LocalToEpoch function. }

Uses oldlinux;

Var year, month, day, hour, minute, second : Word;

```

begin
  Write ('Year      : '); readln(Year);
  Write ('Month     : '); readln(Month);
  Write ('Day        : '); readln(Day);
  Write ('Hour       : '); readln(Hour);
  Write ('Minute    : '); readln(Minute);
  Write ('Seonds   : '); readln(Second);
  Write ('This is  : ');
  Write (LocalToEpoch(year, month, day, hour, minute, second));
  Writeln (' seconds past 00:00 1/1/1980 ');
end.

```

Lstat

Synopsis: Return information about symbolic link. Do not follow the link

Declaration: `function Lstat(Filename: PathStr;var Info: Stat) : Boolean`

Visibility: default

Description: `LStat` gets information about the link specified in `Path`, and stores it in `Info`, which is of type `stat`. Contrary to `FStat`, it stores information about the link, not about the file the link points to. The function returns `True` if the call was succesfull, `False` if the call failed.

Errors: `LinuxError` is used to report errors.

`sys_enoent`Path does not exist.

See also: `FStat` ([816](#)), `FStat` ([815](#))

Listing: `./olinuxex/ex29.pp`

```

program example29;

{ Program to demonstrate the LStat function. }

uses oldlinux;

var f : text;
    i : byte;
    info : stat;

begin
  { Make a file }
  assign (f, 'test.fil');
  rewrite (f);
  for i:=1 to 10 do writeln (f, 'Testline # ', i);
  close (f);
  { Do the call on made file. }
  if not fstat ('test.fil', info) then
    begin
      writeln('Fstat failed. Errno : ', linuxerror);
      halt (1);
    end;
  writeln;
  writeln ('Result of fstat on file ''test.fil''.');
  writeln ('Inode   : ', info.ino);
  writeln ('Mode    : ', info.mode);
  writeln ('nlink   : ', info.nlink);
  writeln ('uid     : ', info.uid);
  writeln ('gid     : ', info.gid);
  writeln ('rdev    : ', info.rdev);
  writeln ('Size    : ', info.size);
  writeln ('Blksize  : ', info.blksize);
  writeln ('Blocks  : ', info.blocks);
  writeln ('atime   : ', info.atime);
  writeln ('mtime   : ', info.mtime);
  writeln ('ctime   : ', info.ctime);

  if not SymLink ('test.fil', 'test.lnk') then
    writeln ('Link failed ! Errno : ', linuxerror);

```

```

if not lstat ('test.lnk',info) then
begin
  writeln('LStat failed. Errno : ',linuxerror);
  halt (1);
end;
writeln;
writeln ('Result of fstat on file ''test.lnk''.');
writeln ('Inode   : ',info.ino);
writeln ('Mode    : ',info.mode);
writeln ('nlink   : ',info.nlink);
writeln ('uid     : ',info.uid);
writeln ('gid     : ',info.gid);
writeln ('rdev    : ',info.rdev);
writeln ('Size    : ',info.size);
writeln ('Blksize  : ',info.blksize);
writeln ('Blocks  : ',info.blocks);
writeln ('atime   : ',info.atime);
writeln ('mtime   : ',info.mtime);
writeln ('ctime   : ',info.ctime);
{ Remove file and link }
erase (f);
unlink ('test.lnk');
end.

```

mkFifo

Synopsis: Create FIFO (named pipe) in file system

Declaration: function mkFifo(pathname: String;mode: LongInt) : Boolean

Visibility: default

Description: MkFifo creates named a named pipe in the filesystem, with name PathName and mode Mode.

Errors: LinuxError is used to report errors:

sys_enfileToo many file descriptors for this process.

sys_enfileThe system file table is full.

See also: POpen ([838](#)), MkFifo ([832](#))

MMap

Synopsis: Create memory map of a file

Declaration: function MMap(const m: tmmmapargs) : LongInt

Visibility: default

Description: MMap maps or unmaps files or devices into memory. The different fields of the argument m determine what and how the mmap maps this:

addressAddress where to mmap the device. This address is a hint, and may not be followed.

sizeSize (in bytes) of area to be mapped.

protProtection of mapped memory. This is a OR-ed combination of the following constants:

PROT_EXECThe memory can be executed.

PROT_READThe memory can be read.

PROT_WRITEThe memory can be written.

PROT_NONEThe memory can not be accessed.

flagsContains some options for the mmap call. It is an OR-ed combination of the following constants:

MAP_FIXEDDo not map at another address than the given address. If the address cannot be used, MMap will fail.

MAP_SHAREDShare this map with other processes that map this object.

MAP_PRIVATECreate a private map with copy-on-write semantics.

MAP_ANONYMOUSfd does not have to be a file descriptor.

One of the options MAP_SHARED and MAP_PRIVATE must be present, but not both at the same time.

fdFile descriptor from which to map.

offsetOffset to be used in file descriptor fd.

The function returns a pointer to the mapped memory, or a -1 in case of an error.

Errors: On error, -1 is returned and LinuxError is set to the error code:

Sys_EBADFfd is not a valid file descriptor and MAP_ANONYMOUS was not specified.

Sys_EACCESSMAP_PRIVATE was specified, but fd is not open for reading. Or MAP_SHARED was asked and PROT_WRITE is set, fd is not open for writing

Sys_EINVALOne of the record fields Start, length or offset is invalid.

Sys_ETXTBUSYMAP_DENYWRITE was set but the object specified by fd is open for writing.

Sys_EAGAINfd is locked, or too much memory is locked.

Sys_ENOMEMNot enough memory for this operation.

See also: MUnMap ([834](#))

Listing: ./olinuxex/ex66.pp

Program Example66;

{ Program to demonstrate the MMap function. }

Uses oldlinux;

Var S : String;

fd, Len : Longint;

args : tmapargs;

P : PChar;

begin

S:= 'This is a string'#0;

Len:=Length(S);

fd:=fdOpen('testfile.txt', Open_wrOnly or open_creat);

If fd=-1 **then**

Halt(1);

If fdWrite(fd, S[1], Len)=-1 **then**

Halt(2);

fdClose(fd);

fdOpen('testfile.txt', Open_rdOnly);

```

    if fd=-1 then
        Halt(3);
        args.address:=0;
        args.offset:=0;
        args.size:=Len+1;
        args.fd:=Fd;
        args.flags:=MAP_PRIVATE;
        args.prot:=PROT_READ or PROT_WRITE;
        P:=Pchar(mmap(args));
        If longint(P)=-1 then
            Halt(4);
        WriteLn('Read in memory : ',P);
        fdclose(fd);
        if Not MUnMap(P,Len) Then
            Halt(LinuxError);
    end.

```

MUnMap

Synopsis: Unmap previously mapped memory block

Declaration: function MUnMap(P: Pointer;Size: LongInt) : Boolean

Visibility: default

Description: MUnMap unmaps the memory block of size Size, pointed to by P, which was previously allocated with MMap (832).

The function returns True if successful, False otherwise.

For an example, see MMap (832).

Errors: In case of error the function returns False and LinuxError is set to an error value. See MMap (832) for possible error values.

See also: MMap (832)

NanoSleep

Synopsis: Suspend process for a short time

Declaration: function NanoSleep(const req: timespec;var rem: timespec) : LongInt

Visibility: default

Description: NanoSleep suspends the process till a time period as specified in req has passed. Then the function returns. If the call was interrupted (e.g. by some signal) then the function may return earlier, and rem will contain the remaining time till the end of the intended period. In this case the return value will be -1, and LinuxError will be set to EINTR

If the function returns without error, the return value is zero.

Errors: If the call was interrupted, -1 is returned, and LinuxError is set to EINTR. If invalid time values were specified, then -1 is returned and LinuxError is set to EINVAL.

See also: Pause (837), Alarm (788)

Listing: ./olinuxex/ex72.pp

```

program example72;

{ Program to demonstrate the NanoSleep function. }

uses oldlinux;

Var
  Req,Rem : TimeSpec;
  Res : Longint;

begin
  With Req do
    begin
      tv_sec:=10;
      tv_nsec:=100;
    end;
  Write( 'NanoSleep returned : ');
  Flush(Output);
  Res:=(NanoSleep(Req,rem));
  Writeln(res);
  If (res<>0) then
    With rem do
      begin
        Writeln( 'Remaining seconds      : ',tv_sec);
        Writeln( 'Remaining nanoseconds : ',tv_nsec);
      end;
end.

```

Nice

Synopsis: Set process priority

Declaration: `procedure Nice(N: Integer)`

Visibility: default

Description: `Nice` adds `-N` to the priority of the running process. The lower the priority numerically, the less the process is favored. Only the superuser can specify a negative `N`, i.e. increase the rate at which the process is run.

Errors: Errors are returned in `LinuxError`

sys_epermA non-superuser tried to specify a negative `N`, i.e. do a priority increase.

See also: `GetPriority` ([824](#)), `SetPriority` ([843](#))

Listing: `./olinuxex/ex15.pp`

```

Program Example15;

{ Program to demonstrate the Nice and Get/SetPriority functions. }

Uses oldlinux;

begin
  writeln ( 'Setting priority to 5 ');
  setpriority ( prio_process , getpid , 5 );

```

```

writeln ( 'New priority = ', getpriority (prio_process, getpid));
writeln ( 'Doing nice 10 ');
nice (10);
writeln ( 'New Priority = ', getpriority (prio_process, getpid));
end.

```

Octal

Synopsis: Convert octal to decimal value

Declaration: `function Octal(l: LongInt) : LongInt`

Visibility: default

Description: `Octal` will convert a number specified as an octal number to it's decimal value.

This is useful for the `Chmod` ([792](#)) call, where permissions are specified as octal numbers.

Errors: No checking is performed whether the given number is a correct Octal number. e.g. specifying 998 is possible; the result will be wrong in that case.

See also: `Chmod` ([792](#))

Listing: `./olinuxex/ex68.pp`

Program `Example68;`

{ Program to demonstrate the Octal function. }

Uses `oldlinux;`

```

begin
  WriteLn ( 'Mode 777 : ', Octal(777));
  WriteLn ( 'Mode 644 : ', Octal(644));
  WriteLn ( 'Mode 755 : ', Octal(755));
end.

```

OpenDir

Synopsis: Open directory for reading

Declaration: `function OpenDir(f: pchar) : PDir`
`function OpenDir(f: String) : PDir`

Visibility: default

Description: `OpenDir` opens the directory `f`, and returns a `pdir` pointer to a `Dir` record, which can be used to read the directory structure. If the directory cannot be opened, `nil` is returned.

Errors: Errors are returned in `LinuxError`.

See also: `CloseDir` ([797](#)), `ReadDir` ([839](#)), `SeekDir` ([841](#)), `TellDir` ([857](#))

Listing: `./olinuxex/ex35.pp`

Program Example35;

```
{ Program to demonstrate the
  OpenDir, ReadDir, SeekDir and Telldir functions. }
```

```
Uses oldlinux;
```

```
Var TheDir : PDir;
     ADirent : PDirent;
     Entry : Longint;
```

```
begin
  TheDir:=OpenDir( ' ./.' );
  Repeat
    Entry:=Telldir(TheDir);
    ADirent:=ReadDir ( TheDir);
    If ADirent<>Nil then
      With ADirent^ do
        begin
          Writeln ( 'Entry No : ',Entry);
          Writeln ( 'Inode   : ',ino);
          Writeln ( 'Offset   : ',off);
          Writeln ( 'Reclen   : ',reclen);
          Writeln ( 'Name     : ',pchar(@name[0]));
        end;
      Until ADirent=Nil;
    Repeat
      Write ( 'Entry No. you would like to see again (-1 to stop): ');
      ReadLn ( Entry);
      If Entry<>-1 then
        begin
          SeekDir ( TheDir,Entry);
          ADirent:=ReadDir ( TheDir);
          If ADirent<>Nil then
            With ADirent^ do
              begin
                Writeln ( 'Entry No : ',Entry);
                Writeln ( 'Inode   : ',ino);
                Writeln ( 'Offset   : ',off);
                Writeln ( 'Reclen   : ',reclen);
                Writeln ( 'Name     : ',pchar(@name[0]));
              end;
            end;
          Until Entry=-1;
        CloseDir ( TheDir);
      end.


---


```

Pause

Synopsis: Wait for a signal

Declaration: procedure Pause

Visibility: default

Description: Pause puts the process to sleep and waits until the application receives a signal. If a signal handler is installed for the received sigal, the handler will be called and after that pause will return control to

the process.

For an example, see Alarm (788).

PClose

Synopsis: Close file opened with POpen (838)

Declaration: `function PClose(var F: text) : LongInt`
`function PClose(var F: file) : LongInt`

Visibility: default

Description: PClose closes a file opened with POpen (838). It waits for the command to complete, and then returns the exit status of the command.

For an example, see POpen (838)

Errors: `LinuxError` is used to report errors. If it is different from zero, the exit status is not valid.

See also: POpen (838)

POpen

Synopsis: Pipe file to standard input/output of program

Declaration: `procedure POpen(var F: text; const Prog: String; rw: Char)`
`procedure POpen(var F: file; const Prog: String; rw: Char)`

Visibility: default

Description: POpen runs the command specified in `Cmd`, and redirects the standard in or output of the command to the other end of the pipe `F`. The parameter `rw` indicates the direction of the pipe. If it is set to 'W', then `F` can be used to write data, which will then be read by the command from `stdin`. If it is set to 'R', then the standard output of the command can be read from `F`. `F` should be reset or rewritten prior to using it. `F` can be of type `Text` or `File`. A file opened with POpen can be closed with `Close`, but also with `PClose` (838). The result is the same, but `PClose` returns the exit status of the command `Cmd`.

Errors: Errors are reported in `LinuxError` and are essentially those of the `Execve`, `Dup` and `AssignPipe` commands.

See also: `AssignPipe` (789), `PClose` (838)

Listing: `./olinuxex/ex37.pp`

Program Example37;

{ Program to demonstrate the Popen function. }

uses oldlinux;

var f : text;
 i : longint;

begin

writeln ('Creating a shell script to which echoes its arguments');
writeln ('and input back to stdout');
assign (f, 'test21a');

```

rewrite (f);
writeln (f, '#!/bin/sh');
writeln (f, 'echo this is the child speaking.... ');
writeln (f, 'echo got arguments \'$*\'' );
writeln (f, 'cat');
writeln (f, 'exit 2');
writeln (f);
close (f);
chmod ('test21a', octal (755));
popen (f, './test21a arg1 arg2', 'W');
if linuxerror <> 0 then
    writeln ('error from POpen : Linuxerror : ', Linuxerror);
for i:=1 to 10 do
    writeln (f, 'This is written to the pipe, and should appear on stdout. ');
Flush(f);
Writeln ('The script exited with status : ', PClose (f));
writeln;
writeln ('Press <return> to remove shell script. ');
readln;
assign (f, 'test21a');
erase (f)
end.

```

ReadDir

Synopsis: Read entry from directory

Declaration: function ReadDir(p: PDir) : pdirent

Visibility: default

Description: ReadDir reads the next entry in the directory pointed to by p. It returns a pdirent pointer to a structure describing the entry. If the next entry can't be read, Nil is returned.

For an example, see OpenDir ([836](#)).

Errors: Errors are returned in LinuxError.

See also: CloseDir ([797](#)), OpenDir ([836](#)), SeekDir ([841](#)), TellDir ([857](#))

ReadLink

Synopsis: Read destination of symbolic link

Declaration: function ReadLink(name: pchar; linkname: pchar; maxlen: LongInt) : LongInt
function ReadLink(name: PathStr) : PathStr

Visibility: default

Description: ReadLink returns the file the symbolic link name is pointing to. The first form of this function accepts a buffer linkname of length maxlen where the filename will be stored. It returns the actual number of characters stored in the buffer.

The second form of the function returns simply the name of the file.

Errors: On error, the first form of the function returns -1; the second one returns an empty string. LinuxError is set to report errors:

SYS_ENOTDIR part of the path in Name is not a directory.

SYS_EINVALmaxlen is not positive, or the file is not a symbolic link.

SYS_ENAMETOOLONGA pathname, or a component of a pathname, was too long.

SYS_ENOENTthe link name does not exist.

SYS_EACCESNo permission to search a directory in the path

SYS_ELOOPToo many symbolic links were encountered in translating the pathname.

SYS_EIOAn I/O error occurred while reading from the file system.

SYS_EFAULTThe buffer is not part of the the process's memory space.

SYS_ENOMEMNot enough kernel memory was available.

See also: SymLink ([849](#))

Listing: ./olinuxex/ex62.pp

Program Example62;

{ Program to demonstrate the ReadLink function. }

Uses oldlinux;

Var F : Text;
S : **String**;

begin
Assign (F, 'test.txt');
Rewrite (F);
WriteLn (F, 'This is written to test.txt');
Close(f);
{ new.txt and test.txt are now the same file }
if not SymLink ('test.txt', 'new.txt') **then**
 writeln ('Error when symlinking !');
S:=ReadLink('new.txt');
If S='' **then**
 WriteLn ('Error reading link !')
Else
 WriteLn ('Link points to : ',S);
{ Now remove links }
If not Unlink ('new.txt') **then**
 WriteLn ('Error when unlinking !');
If not Unlink ('test.txt') **then**
 WriteLn ('Error when unlinking !');
end.

ReadTimezoneFile

Synopsis: Read the timezone file and initialize time routines

Declaration: procedure ReadTimezoneFile(fn: String)

Visibility: default

Description: ReadTimezoneFile reads the timezone file fn and initializes the local time routines based on the information found there.

There should be no need to call this function. The initialization routines of the linux unit call this routine at unit startup.

Errors: None.

See also: [GetTimezoneFile \(825\)](#), [GetLocalTimezone \(822\)](#)

SeekDir

Synopsis: Seek to position in directory

Declaration: `procedure SeekDir(p: PDir; off: LongInt)`

Visibility: default

Description: `SeekDir` sets the directory pointer to the `off`-th entry in the directory structure pointed to by `p`.

For an example, see [OpenDir \(836\)](#).

Errors: Errors are returned in `LinuxError`.

See also: [CloseDir \(797\)](#), [ReadDir \(839\)](#), [OpenDir \(836\)](#), [TellDir \(857\)](#)

Select

Synopsis: Wait for events on file descriptors

Declaration: `function Select(N: LongInt; readfds: pfdset; writefds: pfdset;
exceptfds: pfdset; Timeout: ptimeval) : LongInt`
`function Select(N: LongInt; readfds: pfdset; writefds: pfdset;
exceptfds: pfdset; Timeout: LongInt) : LongInt`

Visibility: default

Description: `Select` checks one of the file descriptors in the `FDSet`s to see if its status changed.

`readfds`, `writefds` and `exceptfds` are pointers to arrays of 256 bits. If you want a file descriptor to be checked, you set the corresponding element in the array to 1. The other elements in the array must be set to zero. Three arrays are passed : The entries in `readfds` are checked to see if characters become available for reading. The entries in `writefds` are checked to see if it is OK to write to them, while entries in `exceptfds` are checked to see if an exception occurred on them.

You can use the functions [FD_ZERO \(811\)](#), [FD_Clr \(810\)](#), [FD_Set \(811\)](#) or [FD_IsSet \(811\)](#) to manipulate the individual elements of a set.

The pointers can be `Nil`.

`N` is the largest index of a nonzero entry plus 1. (= the largest file-descriptor + 1).

`Timeout` can be used to set a time limit. If `Timeout` can be two types :

1. `Timeout` is of type `PTime` and contains a zero time, the call returns immediately. If `Timeout` is `Nil`, the kernel will wait forever, or until a status changed.
2. `Timeout` is of type `Longint`. If it is -1, this has the same effect as a `Timeout` of type `PTime` which is `Nil`. Otherwise, `Timeout` contains a time in milliseconds.

When the `Timeout` is reached, or one of the file descriptors has changed, the `Select` call returns. On return, it will have modified the entries in the array which have actually changed, and it returns the number of entries that have been changed. If the timeout was reached, and no descriptor changed, zero is returned; The arrays of indexes are undefined after that. On error, -1 is returned.

Errors: On error, the function returns -1, and Errors are reported in `LinuxError` :

SYS_EBADF An invalid descriptor was specified in one of the sets.

SYS_EINTR non blocked signal was caught.

SYS_EINVAL is negative or too big.

SYS_ENOMEM `Select` was unable to allocate memory for its internal tables.

See also: `SelectText` (842), `GetFS` (821), `FD_ZERO` (811), `FD_Clr` (810), `FD_Set` (811), `FD_IsSet` (811)

Listing: `./olinuxex/ex33.pp`

Program `Example33`;

{ Program to demonstrate the Select function. }

Uses `oldlinux`;

Var `FDS` : `FDSet`;

begin

`FD_Zero (FDS);`

`FD_Set (0,FDS);`

`Writeln ('Press the <ENTER> to continue the program. ');`

{ Wait until File descriptor 0 (=Input) changes }

`Select (1,@FDS,nil,nil,nil);`

{ Get rid of <ENTER> in buffer }

`readln;`

`Writeln ('Press <ENTER> key in less than 2 seconds... ');`

`FD_Zero (FDS);`

`FD_Set (0,FDS);`

`if Select (1,@FDS,nil,nil,2000)>0 then`

`Writeln ('Thank you ! ')`

{ FD_ISSET(0,FDS) would be true here. }

`else`

`Writeln ('Too late ! ');`

`end.`

SelectText

Synopsis: Wait for event on typed ontyped file.

Declaration: `function SelectText(var T: Text;Timeout: ptimeval) : LongInt`
`function SelectText(var T: Text;Timeout: LongInt) : LongInt`

Visibility: `default`

Description: `SelectText` executes the `Select` (841) call on a file of type `Text`. You can specify a timeout in `Timeout`. The `SelectText` call determines itself whether it should check for read or write, depending on how the file was opened : With `Reset` it is checked for reading, with `Rewrite` and `Append` it is checked for writing.

Errors: See `Select` (841). `SYS_EBADF` can also mean that the file wasn't opened.

See also: `Select` (841), `GetFS` (821)

SetDate

Synopsis: Set the current system date.

Declaration: `function SetDate(Year: Word;Month: Word;Day: Word) : Boolean`

Visibility: default

Description: `SetDate` sets the system date to year, month, day. This is the kernel date, so it is in GMT. The time is not touched. The function returns `True` if the call was executed correctly, `False` otherwise.

Remark: You must be root to execute this call.

Errors: Errors are returned in `LinuxError` (787)

See also: `GetDate` (817), `SetTime` (844), `SetDateTime` (843)

SetDateTime

Synopsis: Set the current system date and time

Declaration: `function SetDateTime(Year: Word;Month: Word;Day: Word;hour: Word;minute: Word;second: Word) : Boolean`

Visibility: default

Description: `SetDate` sets the system date and time to year, month, day, hour, min, Sec. This is the kernel date/time, so it is in GMT. The time is not touched. The function returns `True` if the call was executed correctly, `False` otherwise.

Remark: You must be root to execute this call.

Errors: Errors are returned in `LinuxError` (787)

See also: `SetDate` (842), `SetTime` (844), `GetDateTime` (818)

SetPriority

Synopsis: Set process priority

Declaration: `procedure SetPriority(Which: Integer;Who: Integer;What: Integer)`

Visibility: default

Description: `SetPriority` sets the priority with which a process is running. Which process(es) is determined by the `Which` and `Who` variables. `Which` can be one of the pre-defined constants:

Prio_Process`Who` is interpreted as process ID

Prio_PGrp`Who` is interpreted as process group ID

Prio_User`Who` is interpreted as user ID

`Prio` is a value in the range -20 to 20.

For an example, see `Nice` (835).

Errors: Error checking must be done on `LinuxError`, since a priority can be negative.

sys_esrchNo process found using `which` and `who`.

sys_einval`Which` was not one of `Prio_Process`, `Prio_Grp` or `Prio_User`.

sys_epermA process was found, but neither its effective or real user ID match the effective user ID of the caller.

sys_eaccessA non-superuser tried to a priority increase.

See also: `GetPriority` (824), `Nice` (835)

SetTime

Synopsis: Set the current system time.

Declaration: `function SetTime(Hour: Word;Min: Word;Sec: Word) : Boolean`

Visibility: default

Description: `SetTime` sets the system time to hour, min, Sec. This is the kernel time, so it is in GMT. The date is not touched. The function returns `True` if the call was executed corretly, `False` otherwise.

Remark: You must be root to execute this call.

Errors: Errors are returned in `LinuxError` ([787](#))

See also: `GetTime` ([824](#)), `SetDate` ([842](#)), `SetDateTime` ([843](#))

Shell

Synopsis: Execute and feed command to system shell

Declaration: `function Shell(const Command: String) : LongInt`
`function Shell(const Command: AnsiString) : LongInt`

Visibility: default

Description: `Shell` invokes the bash shell (`/bin/sh`), and feeds it the command `Command` (using the `-c` option). The function then waits for the command to complete, and then returns the exit status of the command, or 127 if it could not complete the `Fork` ([813](#)) or `Execve` ([804](#)) calls.

Errors: Errors are reported in `LinuxError`.

See also: `POpen` ([838](#)), `Fork` ([813](#)), `Execve` ([804](#))

Listing: `./olinuxex/ex56.pp`

```

program example56;

uses oldlinux;

{ Program to demonstrate the Shell function }

Var S : Longint;

begin
  WriteLn ( 'Output of ls -l *.pp' );
  S:= Shell ( 'ls -l *.pp' );
  WriteLn ( 'Command exited wwith status : ',S);
end.

```

SigAction

Synopsis: Install signal handler

Declaration: `procedure SigAction(Signum: LongInt;Act: PSigActionRec;`
`OldAct: PSigActionRec)`

Visibility: default

Description: Changes the action to take upon receipt of a signal. `Act` and `Oldact` are pointers to a `SigActionRec` record. `SigNum` specifies the signal, and can be any signal except **SIGKILL** or **SIGSTOP**.

If `Act` is non-nil, then the new action for signal `SigNum` is taken from it. If `OldAct` is non-nil, the old action is stored there. `Sa_Handler` may be `SIG_DFL` for the default action or `SIG_IGN` to ignore the signal. `Sa_Mask` Specifies which signals should be ignored during the execution of the signal handler. `Sa_Flags` Specifies a series of flags which modify the behaviour of the signal handler. You can 'or' none or more of the following :

SA_NOCLDSTOP If `signum` is **SIGCHLD** do not receive notification when child processes stop.

SA_ONESHOT or **SA_RESETHAND** Restore the signal action to the default state once the signal handler has been called.

SA_RESTART For compatibility with BSD signals.

SA_NOMASK or **SA_NODEFER** Do not prevent the signal from being received from within its own signal handler.

Errors: `LinuxError` is used to report errors.

sys_einval an invalid signal was specified, or it was **SIGKILL** or **SIGSTOP**.

sys_efault `Act`, `OldAct` point outside this process address space

sys_eintr System call was interrupted.

See also: `SigProcMask` (847), `SigPending` (847), `SigSuspend` (848), `Kill` (828)

Listing: `./olinuxex/ex57.pp`

Program `example57`;

```
{ Program to demonstrate the SigAction function.}

{
do a kill -USR1 pid from another terminal to see what happens.
replace pid with the real pid of this program.
You can get this pid by running 'ps'.
}
```

uses `oldlinux`;

Var

`oa, na : PSigActionRec;`

Procedure `DoSig(sig : Longint); cdecl;`

begin

`writeln('Receiving signal: ', sig);`

end;

begin

```
new(na);
new(oa);
na^.Handler.sh:=@DoSig;
na^.Sa_Mask:=0;
na^.Sa_Flags:=0;
na^.Sa_Restorer:=Nil;
SigAction(SigUsr1, na, oa);
if LinuxError <> 0 then
```

```

    begin
    writeln('Error: ',linuxerror, '.');
    halt(1);
    end;
    Writeln('Send USR1 signal or press <ENTER> to exit');
    readln;
end.

```

Signal

Synopsis: Install signal handler (deprecated)

Declaration: function Signal(Signum: LongInt;Handler: SignalHandler) : SignalHandler

Visibility: default

Description: Signal installs a new signal handler for signal SigNum. This call has the same functionality as the **SigAction** call. The return value for Signal is the old signal handler, or nil on error.

Errors: LinuxError is used to report errors :

SIG_ERRAn error occurred.

See also: SigAction ([844](#)), Kill ([828](#))

Listing: ./olinuxex/ex58.pp

Program example58;

{ Program to demonstrate the Signal function. }

```

{
do a kill -USR1 pid from another terminal to see what happens.
replace pid with the real pid of this program.
You can get this pid by running 'ps'.
}

```

uses oldlinux;

Procedure DoSig(sig : Longint);**cdecl**;

```

begin
    writeln('Receiving signal: ',sig);
end;

begin
    SigNal(SigUsr1, @DoSig);
    if LinuxError <> 0 then
    begin
        writeln('Error: ',linuxerror, '.');
        halt(1);
    end;
    Writeln('Send USR1 signal or press <ENTER> to exit');
    readln;
end.

```

SigPending

Synopsis: Return set of currently pending signals

Declaration: `function SigPending : SigSet`

Visibility: default

Description: Sigpending allows the examination of pending signals (which have been raised while blocked.) The signal mask of pending signals is returned.

Errors: None

See also: SigAction (844), SigProcMask (847), SigSuspend (848), Signal (846), Kill (828)

SigProcMask

Synopsis: Set list of blocked signals

Declaration: `procedure SigProcMask(How: LongInt; SSet: PSigSet; OldSSet: PSigSet)`

Visibility: default

Description: Changes the list of currently blocked signals. The behaviour of the call depends on How :

SIG_BLOCKThe set of blocked signals is the union of the current set and the SSet argument.

SIG_UNBLOCKThe signals in SSet are removed from the set of currently blocked signals.

SIG_SETMASKThe list of blocked signals is set so SSet.

If OldSSet is non-nil, then the old set is stored in it.

Errors: `LinuxError` is used to report errors.

sys_efaultSSet or OldSSet point to an adress outside the range of the process.

sys_eintrSystem call was interrupted.

See also: SigAction (844), SigPending (847), SigSuspend (848), Kill (828)

SigRaise

Synopsis: Raise a signal (send to current process)

Declaration: `procedure SigRaise(Sig: Integer)`

Visibility: default

Description: SigRaise sends a Sig signal to the current process.

Errors: None.

See also: Kill (828), GetPid (823)

Listing: ./olinuxex/ex65.pp

```

Program example64;

{ Program to demonstrate the SigRaise function.}

uses oldlinux;

Var
    oa, na : PSigActionRec;

Procedure DoSig(sig : Longint); cdecl;

begin
    writeln( 'Receiving signal: ', sig);
end;

begin
    new(na);
    new(oa);
    na^.handler.sh:=@DoSig;
    na^.Sa_Mask:=0;
    na^.Sa_Flags:=0;
    na^.Sa_Restorer:=Nil;
    SigAction( SigUshr1, na, oa);
    if LinuxError <> 0 then
        begin
            writeln( 'Error: ', linuxerror, '. ');
            halt(1);
        end;
    Writeln( 'Sending USR1 ( ', sigusr1, ') signal to self. ');
    SigRaise(sigusr1);
end.

```

SigSuspend

Synopsis: Set signal mask and suspend process till signal is received

Declaration: `procedure SigSuspend(Mask: SigSet)`

Visibility: default

Description: SigSuspend temporarily replaces the signal mask for the process with the one given in Mask, and then suspends the process until a signal is received.

Errors: None

See also: SigAction ([844](#)), SigProcMask ([847](#)), SigPending ([847](#)), Signal ([846](#)), Kill ([828](#))

StringToPPChar

Synopsis: Split string in list of null-terminated strings

Declaration: `function StringToPPChar(var S: String) : ppchar`
`function StringToPPChar(var S: AnsiString) : ppchar`
`function StringToPPChar(S: Pchar) : ppchar`

Visibility: default

Description: `StringToPPChar` splits the string `S` in words, replacing any whitespace with zero characters. It returns a pointer to an array of `pchars` that point to the first letters of the words in `S`. This array is terminated by a `Nil` pointer.

The function does *not* add a zero character to the end of the string unless it ends on whitespace.

The function reserves memory on the heap to store the array of `PChar`; The caller is responsible for freeing this memory.

This function can be called to create arguments for the various `Exec` calls.

Errors: None.

See also: `CreateShellArgV` ([797](#)), `Execve` ([804](#)), `Execv` ([803](#))

Listing: `./olinuxex/ex70.pp`

Program `Example70`;

{ Program to demonstrate the StringToPPchar function. }

Uses `oldlinux`;

Var `S : String`;
 `P : PPChar`;
 `I : longint`;

begin
 // remark whitespace at end.
 `S := 'This is a string with words. '`;
 `P := StringToPPChar(S)`;
 `I := 0`;
 While `P[I] <> Nil` **do**
 begin
 `Writeln('Word ', I, ' : ', P[I])`;
 `Inc(I)`;
 end;
 `FreeMem(P, I * SizeOf(Pchar))`;
end.

SymLink

Synopsis: Create a symbolic link

Declaration: `function SymLink(OldPath: PathStr; NewPath: PathStr) : Boolean`

Visibility: `default`

Description: `SymLink` makes `Newpath` point to the file in `OldPath`, which doesn't necessarily exist. The two files DO NOT have the same inode number. This is known as a 'soft' link.

The permissions of the link are irrelevant, as they are not used when following the link. Ownership of the file is only checked in case of removal or renaming of the link.

The function returns `True` if the call was succesfull, `False` if the call failed.

Errors: Errors are returned in `LinuxError`.

sys_eperm The filesystem containing `oldpath` and `newpath` does not support linking files.

sys_eaccess Write access for the directory containing Newpath is disallowed, or one of the directories in OldPath or NewPath has no search (=execute) permission.

sys_enoent A directory entry in OldPath or NewPath does not exist or is a symbolic link pointing to a non-existent directory.

sys_enotdir A directory entry in OldPath or NewPath is not a directory.

sys_enomem Insufficient kernel memory.

sys_erofs The files are on a read-only filesystem.

sys_eexist NewPath already exists.

sys_eloop OldPath or NewPath has a reference to a circular symbolic link, i.e. a symbolic link, whose expansion points to itself.

sys_enospc The device containing NewPath has no room for another entry.

See also: Link (829), UnLink (859), ReadLink (839)

Listing: ./olinuxex/ex22.pp

Program Example22;

{ Program to demonstrate the SymLink and UnLink functions. }

Uses oldlinux;

Var F : Text;
S : String;

begin
Assign (F, 'test.txt');
Rewrite (F);
WriteIn (F, 'This is written to test.txt');
Close(f);
{ new.txt and test.txt are now the same file }
if not SymLink ('test.txt', 'new.txt') **then**
 writeln ('Error when symlinking !');
{ Removing test.txt still leaves new.txt
Pointing now to a non-existent file ! }
If not Unlink ('test.txt') **then**
 Writeln ('Error when unlinking !');
Assign (f, 'new.txt');
{ This should fail, since the symbolic link
points to a non-existent file ! }
{\$i-}
Reset (F);
{\$i+}
If IOResult=0 then
 Writeln ('This shouldn''t happen');
{ Now remove new.txt also }
If not Unlink ('new.txt') **then**
 Writeln ('Error when unlinking !');
end.

SysCall

Synopsis: Execute system call.

Declaration: `function SysCall(callnr: LongInt; var regs: SysCallRegs) : LongInt`

Visibility: default

Description: `SysCall` can be used to execute a direct system call. The call parameters must be encoded in `regs` and the call number must be specified by `callnr`. The call result is returned, and any modified registers are in `regs`

Errors: None.

See also: `SysCallRegs` ([783](#))

Sysinfo

Synopsis: Return kernel system information

Declaration: `function Sysinfo(var Info: TSysinfo) : Boolean`

Visibility: default

Description: `SysInfo` returns system information in `Info`. Returned information in `Info` includes:

uptime Number of seconds since boot.

loads 1, 5 and 15 minute load averages.

totalram total amount of main memory.

freeram amount of free memory.

sharedram amount of shared memory.

bufferram amount of memory used by buffers.

totalswap total amount of swap space.

freeswap amount of free swap space.

procs number of current processes.

Errors: None.

See also: `Uname` ([858](#))

Listing: `./olinuxex/ex64.pp`

program Example64;

{ Example to demonstrate the SysInfo function }

Uses oldlinux;

Function Mb(L : Longint) : longint;

begin

 Mb:=L div (1024*1024);

end;

Var Info : TSysInfo;

 D,M,Secs,H : longint;

begin

If Not SysInfo(Info) **then**

 Halt(1);


```

With Info do
  begin
    D:=Uptime div (3600*24);
    UpTime:=UpTime mod (3600*24);
    h:=uptime div 3600;
    uptime:=uptime mod 3600;
    m:=uptime div 60;
    secs:=uptime mod 60;
    Writeln( 'Uptime : ',d,'days, ',h,' hours, ',m,' min, ',secs,' s. ');
    Writeln( 'Loads   : ',Loads[1],'/ ',Loads[2],'/ ',Loads[3]);
    Writeln( 'Total Ram   : ',Mb(totalram), 'Mb. ');
    Writeln( 'Free Ram    : ',Mb(freeram), 'Mb. ');
    Writeln( 'Shared Ram  : ',Mb(sharedram), 'Mb. ');
    Writeln( 'Buffer Ram  : ',Mb(bufferram), 'Mb. ');
    Writeln( 'Total Swap  : ',Mb(totalswap), 'Mb. ');
    Writeln( 'Free Swap   : ',Mb(freeswap), 'Mb. ');
  end;
end.

```

S_ISBLK

Synopsis: Is file a block device

Declaration: `function S_ISBLK(m: Word) : Boolean`

Visibility: default

Description: `S_ISBLK` checks the file mode `m` to see whether the file is a block device file. If so it returns `True`.

See also: `FStat` ([816](#)), `S_ISLNK` ([853](#)), `S_ISREG` ([854](#)), `S_ISDIR` ([852](#)), `S_ISCHR` ([852](#)), `S_ISFIFO` ([853](#)), `S_ISSOCK` ([854](#))

S_ISCHR

Synopsis: Is file a character device

Declaration: `function S_ISCHR(m: Word) : Boolean`

Visibility: default

Description: `S_ISCHR` checks the file mode `m` to see whether the file is a character device file. If so it returns `True`.

See also: `FStat` ([816](#)), `S_ISLNK` ([853](#)), `S_ISREG` ([854](#)), `S_ISDIR` ([852](#)), `S_ISBLK` ([852](#)), `S_ISFIFO` ([853](#)), `S_ISSOCK` ([854](#))

S_ISDIR

Synopsis: Is file a directory

Declaration: `function S_ISDIR(m: Word) : Boolean`

Visibility: default

Description: `S_ISDIR` checks the file mode `m` to see whether the file is a directory. If so it returns `True`

See also: `FStat` ([816](#)), `S_ISLNK` ([853](#)), `S_ISREG` ([854](#)), `S_ISCHR` ([852](#)), `S_ISBLK` ([852](#)), `S_ISFIFO` ([853](#)), `S_ISSOCK` ([854](#))

S_ISFIFO

Synopsis: Is file a FIFO

Declaration: `function S_ISFIFO(m: Word) : Boolean`

Visibility: default

Description: `S_ISFIFO` checks the file mode `m` to see whether the file is a fifo (a named pipe). If so it returns `True`.

See also: `FStat` (816), `S_ISLNK` (853), `S_ISREG` (854), `S_ISCHR` (852), `S_ISBLK` (852), `S_ISDIR` (852), `S_ISSOCK` (854)

S_ISLNK

Synopsis: Is file a symbolic link

Declaration: `function S_ISLNK(m: Word) : Boolean`

Visibility: default

Description: `S_ISLNK` checks the file mode `m` to see whether the file is a symbolic link. If so it returns `True`

See also: `FStat` (816), `S_ISFIFO` (853), `S_ISREG` (854), `S_ISCHR` (852), `S_ISBLK` (852), `S_ISDIR` (852), `S_ISSOCK` (854)

Listing: `./olinuxex/ex53.pp`

Program `Example53`;

{ Program to demonstrate the S_ISLNK function. }

Uses `oldlinux`;

Var `Info` : `Stat`;

begin

if `LStat (paramstr(1),info)` **then**

begin

if `S_ISLNK(info.mode)` **then**
 WriteLn ('File is a link');

if `S_ISREG(info.mode)` **then**
 WriteLn ('File is a regular file');

if `S_ISDIR(info.mode)` **then**
 WriteLn ('File is a directory');

if `S_ISCHR(info.mode)` **then**
 WriteLn ('File is a character device file');

if `S_ISBLK(info.mode)` **then**
 WriteLn ('File is a block device file');

if `S_ISFIFO(info.mode)` **then**
 WriteLn ('File is a named pipe (FIFO)');

if `S_ISSOCK(info.mode)` **then**
 WriteLn ('File is a socket');

end;

end.

S_ISREG

Synopsis: Is file a regular file

Declaration: `function S_ISREG(m: Word) : Boolean`

Visibility: default

Description: `S_ISREG` checks the file mode `m` to see whether the file is a regular file. If so it returns `True`

See also: `FStat` (816), `S_ISFIFO` (853), `S_ISLNK` (853), `S_ISCHR` (852), `S_ISBLK` (852), `S_ISDIR` (852), `S_ISSOCK` (854)

S_ISSOCK

Synopsis: Is file a unix socket

Declaration: `function S_ISSOCK(m: Word) : Boolean`

Visibility: default

Description: `S_ISSOCK` checks the file mode `m` to see whether the file is a socket. If so it returns `True`.

See also: `FStat` (816), `S_ISFIFO` (853), `S_ISLNK` (853), `S_ISCHR` (852), `S_ISBLK` (852), `S_ISDIR` (852), `S_ISREG` (854)

TCDrain

Synopsis: Terminal control: Wait till all data was transmitted

Declaration: `function TCDrain(fd: LongInt) : Boolean`

Visibility: default

Description: `TCDrain` waits until all data to file descriptor `Fd` is transmitted.

The function returns `True` if the call was succesfull, `False` otherwise.

Errors: Errors are reported in `LinuxError`

See also: `TCFlow` (854), `TCFlush` (855), `TCGetAttr` (855), `TCGetPGrp` (856), `TCSendBreak` (856), `TCSetAttr` (857), `TCSetPGrp` (857), `TTYName` (858), `IsATTY` (828)

TCFlow

Synopsis: Terminal control: Suspend transmission of data

Declaration: `function TCFlow(fd: LongInt;act: LongInt) : Boolean`

Visibility: default

Description: `TCFlow` suspends/resumes transmission or reception of data to or from the file descriptor `Fd`, depending on the action `Act`.

This can be one of the following pre-defined values:

TCOOFFsuspend reception/transmission

TCOONresume reception/transmission

TCIOFFtransmit a stop character to stop input from the terminal

TCIONtransmit start to resume input from the terminal.

The function returns `True` if the call was succesfull, `False` otherwise.

Errors: Errors are reported in `LinuxError`.

See also: [TCDrain \(854\)](#), [TCFlush \(855\)](#), [TCGetAttr \(855\)](#), [TCGetPGrp \(856\)](#), [TCSendBreak \(856\)](#), [TCSetAttr \(857\)](#), [TCSetPGrp \(857\)](#), [TTYName \(858\)](#), [IsATTY \(828\)](#)

TCFlush

Synopsis: Terminal control: Discard data buffer

Declaration: `function TCFlush(fd: LongInt;qsel: LongInt) : Boolean`

Visibility: `default`

Description: `TCFlush` discards all data sent or received to/from file descriptor `fd`. `QSel` indicates which queue should be discard. It can be one of the following pre-defined values :

TCIFLUSHinput buffer

TCOFLUSHoutput buffer

TCIOFLUSHboth input and output buffers

The function returns `True` if the call was succesfull, `False` otherwise.

Errors: Errors are reported in `LinuxError`.

See also: [TCDrain \(854\)](#), [TCFlow \(854\)](#), [TCGetAttr \(855\)](#), [TCGetPGrp \(856\)](#), [TCSendBreak \(856\)](#), [TCSetAttr \(857\)](#), [TCSetPGrp \(857\)](#), [TTYName \(858\)](#), [IsATTY \(828\)](#)

TCGetAttr

Synopsis: Terminal Control: Get terminal attributes

Declaration: `function TCGetAttr(fd: LongInt;var tios: Termios) : Boolean`

Visibility: `default`

Description: `TCGetAttr` gets the terminal parameters from the terminal referred to by the file descriptor `fd` and returns them in a `TermIOS` structure `tios`. The function returns `True` if the call was succesfull, `False` otherwise.

Errors: Errors are reported in `LinuxError`

See also: [TCDrain \(854\)](#), [TCFlow \(854\)](#), [TCFlush \(855\)](#), [TCGetPGrp \(856\)](#), [TCSendBreak \(856\)](#), [TCSetAttr \(857\)](#), [TCSetPGrp \(857\)](#), [TTYName \(858\)](#), [IsATTY \(828\)](#)

Listing: `./olinuxex/ex55.pp`

Program `Example55;`

uses `oldlinux;`

{ Program to demonstrate the TCGetAttr/TCSetAttr/CFMakeRaw functions. }

procedure `ShowTermios(var tios:Termios);`
begin

```

    WriteLn('Input Flags : $',hexstr(tios.c_iflag,8)+#13);
    WriteLn('Output Flags : $',hexstr(tios.c_oflag,8));
    WriteLn('Line Flags : $',hexstr(tios.c_lflag,8));
    WriteLn('Control Flags: $',hexstr(tios.c_cflag,8));
end;

var
    oldios,
    tios : Termios;
begin
    WriteLn('Old attributes:');
    TCGetAttr(1,tios);
    ShowTermios(tios);
    oldios:=tios;
    WriteLn('Setting raw terminal mode');
    CFMakeRaw(tios);
    TCSetAttr(1,TCSANOW,tios);
    WriteLn('Current attributes:');
    TCGetAttr(1,tios);
    ShowTermios(tios);
    TCSetAttr(1,TCSANOW,oldios);
end.

```

TCGetPGrp

Synopsis: Terminal control: Get process group

Declaration: `function TCGetPGrp(fd: LongInt;var id: LongInt) : Boolean`

Visibility: default

Description: TCGetPGrp returns the process group ID of a foreground process group in Id The function returns True if the call was succesfull, False otherwise.

Errors: Errors are reported in LinuxError

See also: TCDrain (854), TCFlow (854), TCFlush (855), TCGetAttr (855), TCSendBreak (856), TCSetAttr (857), TCSetPGrp (857), TTYName (858), IsATTY (828)

TCSendBreak

Synopsis: Terminal control: Send break

Declaration: `function TCSendBreak(fd: LongInt;duration: LongInt) : Boolean`

Visibility: default

Description: TCSendBreak Sends zero-valued bits on an asynchrone serial connection decsribed by file-descriptor fd, for duration Duration. The function returns True if the action was performed successfully, False otherwise.

Errors: Errors are reported in LinuxError.

See also: TCDrain (854), TCFlow (854), TCFlush (855), TCGetAttr (855), TCGetPGrp (856), TCSetAttr (857), TCSetPGrp (857), TTYName (858), IsATTY (828)

TCSetAttr

Synopsis: Terminal control: Set attributes

Declaration: `function TCSetAttr(fd: LongInt; OptAct: LongInt; const tios: Termios)
: Boolean`

Visibility: default

Description: `TCSetAttr` sets the terminal parameters you specify in a `TermIOS` structure `Tios` for the terminal referred to by the file descriptor `Fd`.

`OptAct` specifies an optional action when the set need to be done, this could be one of the following pre-defined values:

TCSANOW set immediately.

TCSADRAIN wait for output.

TCSAFLUSH wait for output and discard all input not yet read.

The function Returns `True` if the call was succesfull, `False` otherwise.

For an example, see `TCGetAttr` (855).

Errors: Errors are reported in `LinuxError`.

See also: `TCDrain` (854), `TCFlow` (854), `TCFlush` (855), `TCGetAttr` (855), `TCGetPGrp` (856), `TCSendBreak` (856), `TCSetPGrp` (857), `TTYName` (858), `IsATTY` (828)

TCSetPGrp

Synopsis: Terminal control: Set process group

Declaration: `function TCSetPGrp(fd: LongInt; id: LongInt) : Boolean`

Visibility: default

Description: `TCSetPGrp` Sets the Process Group Id to `Id`. The function returns `True` if the call was successful, `False` otherwise.

For an example, see `TCGetPGrp` (856).

Errors: Errors are returned in `LinuxError`.

See also: `TCDrain` (854), `TCFlow` (854), `TCFlush` (855), `TCGetAttr` (855), `TCGetPGrp` (856), `TCSendBreak` (856), `TCSetAttr` (857), `TTYName` (858), `IsATTY` (828)

TellDir

Synopsis: Return current location in a directory

Declaration: `function TellDir(p: PDir) : LongInt`

Visibility: default

Description: `TellDir` returns the current location in the directory structure pointed to by `p`. It returns -1 on failure.

For an example, see `OpenDir` (836).

Errors: Errors are returned in `LinuxError`.

See also: `CloseDir` (797), `ReadDir` (839), `SeekDir` (841), `OpenDir` (836)

TTYname

Synopsis: Terminal control: Get terminal name

Declaration: `function TTYname(Handle: LongInt) : String`
`function TTYname(var F: Text) : String`

Visibility: default

Description: TTYNameReturns the name of the terminal pointed to by `f`. `f` must be a terminal. `f` can be of type:

1. `longint` for file handles;
2. `Text` for text variables such as `input` etc.

Errors: Returns an empty string in case of an error. `Linuxerror` may be set to indicate what error occurred, but this is uncertain.

See also: [TCDrain \(854\)](#), [TCFlow \(854\)](#), [TCFlush \(855\)](#), [TCGetAttr \(855\)](#), [TCGetPGrp \(856\)](#), [TCSendBreak \(856\)](#), [TCSetAttr \(857\)](#), [TCSetPGrp \(857\)](#), [IsATTY \(828\)](#), [IOCtl \(827\)](#)

Umask

Synopsis: Set file creation mask.

Declaration: `function Umask(Mask: Integer) : Integer`

Visibility: default

Description: Change the file creation mask for the current user to `Mask`. The current mask is returned.

See also: [Chmod \(792\)](#)

Listing: `./olinuxex/ex27.pp`

Program `Example27;`

{ Program to demonstrate the Umask function. }

Uses `oldlinux;`

begin

WriteLn ('Old Umask was : ', Umask(Octal(111)));

WRiteLn ('New Umask is : ', Octal(111));

end.

Uname

Synopsis: Return system name.

Declaration: `function Uname(var unamerec: utsname) : Boolean`

Visibility: default

Description: Uname gets the name and configuration of the current linux kernel, and returns it in `unamerec`.

Errors: `LinuxError` is used to report errors.

See also: [GetHostName \(822\)](#), [GetDomainName \(818\)](#)

UnLink

Synopsis: Unlink (i.e. remove) a file.

Declaration: `function UnLink(Path: PathStr) : Boolean`
`function UnLink(Path: pchar) : Boolean`

Visibility: default

Description: UnLink decreases the link count on file Path. Path can be of type PathStr or PChar. If the link count is zero, the file is removed from the disk. The function returns True if the call was succesfull, False if the call failed.

For an example, see Link (829).

Errors: Errors are returned in LinuxError.

sys_eaccessYou have no write access right in the directory containing Path, or you have no search permission in one of the directory components of Path.

sys_epermThe directory containing pathname has the sticky-bit set and the process's effective uid is neither the uid of the file to be deleted nor that of the directory containing it.

sys_enoentA component of the path doesn't exist.

sys_enotdirA directory component of the path is not a directory.

sys_eisdirPath refers to a directory.

sys_enomemInsufficient kernel memory.

sys_erofsPath is on a read-only filesystem.

See also: Link (829), SymLink (849)

Utime

Synopsis: Set access and modification times of a file (touch).

Declaration: `function Utime(const path: PathStr; utim: UTimeBuf) : Boolean`

Visibility: default

Description: Utime sets the access and modification times of a file. the utimbuf record contains 2 fields, actime, and modtime, both of type Longint. They should be filled with an epoch-like time, specifying, respectively, the last access time, and the last modification time. For some filesystem (most notably, FAT), these times are the same.

Errors: Errors are returned in LinuxError.

sys_eaccessOne of the directories in Path has no search (=execute) permission.

sys_enoentA directory entry in Path does not exist or is a symbolic link pointing to a non-existent directory.

Other errors may occur, but aren't documented.

See also: GetEpochTime (820), Chown (794), Access (787)

Listing: ./olinuxex/ex25.pp

Program Example25;

```
{ Program to demonstrate the UTime function. }
```

Uses oldlinux;

```
Var utim : utimbuf;  
      year,month,day,hour ,minute ,second : Word;
```

```

begin
  { Set access and modification time of executable source }
  GetTime (hour,minute,second);
  GetDate (year,month,day);
  utim.actime:=LocalToEpoch(year,month,day,hour,minute,second);
  utim.modtime:=utim.actime;
  if not Utime('ex25.pp',utim) then
    writeln ('Call to UTime failed !')
  else
    begin
      Write ('Set access and modification times to : ');
      Write (Hour:2,':',minute:2,':',second,',',');
      Writeln (Day:2,'/',month:2,'/',year:4);
    end;
end.

```

WaitPid

: Wait for a process to terminate

```

: function WaitPid(Pid: LongInt; Status: pointer; Options: LongInt)
      : LongInt

```

: default

`WaitPid` waits for a child process with process ID `Pid` to exit. The value of `Pid` can be one of the following:

Pid < -1 Causes `WaitPid` to wait for any child process whose process group ID equals the absolute value of `pid`.

Pid = -1 Causes WaitPid to wait for any child process.

Pid = 0 Causes `WaitPid` to wait for any child process whose process group ID equals the one of the calling process.

Pid > 0 Causes `waitPid` to wait for the child whose process ID equals the value of `Pid`.

The `Options` parameter can be used to specify further how `WaitPid` behaves:

WNOHANG Causes Waitpid to return immediately if no child has exited.

WUNTRACED causes `WaitPid` to return also for children which are stopped, but whose status has not yet been reported.

__WCLONE Causes WaitPid also to wait for threads created by the Clone (795) call.

Upon return, it returns the exit status of the process, or -1 in case of failure.

For an example, see Fork (813).

: Errors are returned in `LinuxError`.

: Fork (813), Execve (804)

WaitProcess

Synopsis: Wait for process to terminate.

Declaration: `function WaitProcess(Pid: LongInt) : LongInt`

Visibility: default

Description: `WaitProcess` waits for process `PID` to exit. `WaitProcess` is equivalent to the `WaitPID` (860) call:

```
WaitPid(PID,@result,0)
```

Handles of Signal interrupts (`errno=EINTR`), and returns the Exitcode of Process `PID` (≥ 0) or - Status if it was terminated

Errors: None.

See also: `WaitPID` (860), `WTERMSIG` (862), `WSTOPSIG` (862), `WIFEXITED` (861), `WIFSTOPPED` (862), `WIFSIGNALED` (861), `W_EXITCODE` (862), `W_STOPCODE` (863), `WEXITSTATUS` (861)

WEXITSTATUS

Synopsis: Extract the exit status from the `WaitPID` (860) result.

Declaration: `function WEXITSTATUS(Status: Integer) : Integer`

Visibility: default

Description: `WEXITSTATUS` can be used to extract the exit status from `Status`, the result of the `WaitPID` (860) call.

See also: `WaitPID` (860), `WaitProcess` (861), `WTERMSIG` (862), `WSTOPSIG` (862), `WIFEXITED` (861), `WIFSTOPPED` (862), `WIFSIGNALED` (861), `W_EXITCODE` (862), `W_STOPCODE` (863)

WIFEXITED

Synopsis: Check whether the process exited normally

Declaration: `function WIFEXITED(Status: Integer) : Boolean`

Visibility: default

Description: `WIFEXITED` checks `Status` and returns `True` if the status indicates that the process terminated normally, i.e. was not stopped by a signal.

See also: `WaitPID` (860), `WaitProcess` (861), `WTERMSIG` (862), `WSTOPSIG` (862), `WIFSTOPPED` (862), `WIFSIGNALED` (861), `W_EXITCODE` (862), `W_STOPCODE` (863), `WEXITSTATUS` (861)

WIFSIGNALED

Synopsis: Check whether the process was exited by a signal.

Declaration: `function WIFSIGNALED(Status: Integer) : Boolean`

Visibility: default

Description: `WIFSIGNALED` returns `True` if `Status` indicates that the process exited because it received a signal.

See also: `WaitPID` (860), `WaitProcess` (861), `WTERMSIG` (862), `WSTOPSIG` (862), `WIFEXITED` (861), `WIFSTOPPED` (862), `W_EXITCODE` (862), `W_STOPCODE` (863), `WEXITSTATUS` (861)

WIFSTOPPED

Synopsis: Check whether the process is currently stopped.

Declaration: `function WIFSTOPPED(Status: Integer) : Boolean`

Visibility: default

Description: WIFSTOPPED checks Status and returns true if the process is currently stopped. This is only possible if WUNTRACED was specified in the options of WaitPID (860).

See also: WaitPID (860), WaitProcess (861), WTERMSIG (862), WSTOPSIG (862), WIFEXITED (861), WIFSIGNALED (861), W_EXITCODE (862), W_STOPCODE (863), WEXITSTATUS (861)

WSTOPSIG

Synopsis: Return the exit code from the process.

Declaration: `function WSTOPSIG(Status: Integer) : Integer`

Visibility: default

Description: WSTOPSIG is an alias for WEXITSTATUS (861).

See also: WaitPID (860), WaitProcess (861), WTERMSIG (862), WIFEXITED (861), WIFSTOPPED (862), WIFSIGNALED (861), W_EXITCODE (862), W_STOPCODE (863), WEXITSTATUS (861)

WTERMSIG

Synopsis: Return the signal that caused a process to exit.

Declaration: `function WTERMSIG(Status: Integer) : Integer`

Visibility: default

Description: WTERMSIG extracts from Status the signal number which caused the process to exit.

See also: WaitPID (860), WaitProcess (861), WSTOPSIG (862), WIFEXITED (861), WIFSTOPPED (862), WIFSIGNALED (861), W_EXITCODE (862), W_STOPCODE (863), WEXITSTATUS (861)

W_EXITCODE

Synopsis: Construct an exit status based on an return code and signal.

Declaration: `function W_EXITCODE(ReturnCode: Integer;Signal: Integer) : Integer`

Visibility: default

Description: W_EXITCODE combines ReturnCode and Signal to a status code fit for WaitPid.

See also: WaitPID (860), WaitProcess (861), WTERMSIG (862), WSTOPSIG (862), WIFEXITED (861), WIFSTOPPED (862), WIFSIGNALED (861), W_STOPCODE (863), WEXITSTATUS (861)

W_STOPCODE

Synopsis: Construct an exit status based on a signal.

Declaration: `function W_STOPCODE(Signal: Integer) : Integer`

Visibility: `default`

Description: `W_STOPCODE` constructs an exit status based on `Signal`, which will cause `WIFSIGNALED` (861) to return `True`

See also: `WaitPID` (860), `WaitProcess` (861), `WTERMSIG` (862), `WSTOPSIG` (862), `WIFEXITED` (861), `WIFSTOPPED` (862), `WIFSIGNALED` (861), `W_EXITCODE` (862), `WEXITSTATUS` (861)

Chapter 23

Reference for unit 'ports'

23.1 Overview

The ports unit implements the `port` constructs found in Turbo Pascal. It uses classes and default array properties to do this.

The unit exists on linux, os/2 and dos. It is implemented only for compatibility with Turbo Pascal. It's usage is discouraged, because using ports is not portable programming, and the operating system may not even allow it (for instance Windows).

Under linux, your program must be run as root, or the `IOPerm` call must be set in order to set appropriate permissions on the port access.

23.2 Constants, types and variables

Variables

`port : tport`

Default instance of type `TPort` ([865](#)). Do not free. This variable is initialized in the unit initialization code, and freed at finalization.

Since there is a default property for a variable of this type, a sentence as

```
port[221]:=12;
```

Will result in the integer 12 being written to port 221, if port is defined as a variable of type `tport`

`portb : tport`

Default instance of type `TPort` ([865](#)). Do not free. This variable is initialized in the unit initialization code, and freed at finalization.

Since there is a default property for a variable of this type, a sentence as

```
portb[221]:=12;
```

Will result in the byte 12 being written to port 221, if port is defined as a variable of type `tport`

```
portl : tportl
```

Default instance of type TPortL ([866](#)). Do not free. This variable is initialized in the unit initialization code, and freed at finalization.

Since there is a default property for a variable of this type, a sentence as

```
portl[221]:=12;
```

Will result in the longint 12 being written to port 221, if port is defined as a variable of type tport

```
portw : tportw
```

Default instance of type TPortW ([867](#)). Do not free. This variable is initialized in the unit initialization code, and freed at finalization.

Since there is a default property for a variable of this type, a sentence as

```
portb[221]:=12;
```

Will result in the word 12 being written to port 221, if port is defined as a variable of type tport

23.3 tport

Description

The TPort type is implemented specially for access to the ports in a TP compatible manner. There is no need to create an instance of this type: the standard TP variables are instantiated at unit initialization.

Method overview

Page	Method	Description
866	readport	Reads an integer of data from a port
865	writeport	Writes an integer of data to a port

Property overview

Page	Property	Access	Description
866	pp	rw	Access integer-sized port by port number

tport.writeport

Synopsis: Writes an integer of data to a port

Declaration: `procedure writeport(p: LongInt; data: Byte)`

Visibility: protected

Description: Writes an integer of data to a port

See also: `tport.readport` ([866](#))

tport.readport

Synopsis: Reads an integer of data from a port

Declaration: `function readport(p: LongInt) : Byte`

Visibility: protected

Description: Reads an integer of data from a port

See also: `tport.writeport` ([865](#))

tport.pp

Synopsis: Access integer-sized port by port number

Declaration: `Property pp[w: LongInt]: Byte; default`

Visibility: public

Access: Read,Write

Description: Access integer-sized port by port number

23.4 tportl**Description**

The `TPortL` type is implemented specially for access to the ports in a TP compatible manner. There is no need to create an instance of this type: the standard TP variables are instantiated at unit initialization.

Method overview

Page	Method	Description
867	<code>readport</code>	Reads a Longint of data from a port
866	<code>writeport</code>	Writes a Longint of data to a port

Property overview

Page	Property	Access	Description
867	<code>pp</code>	<code>rw</code>	Access Longint-sized port by port number

tportl.writeport

Synopsis: Writes a Longint of data to a port

Declaration: `procedure writeport(p: LongInt; data: LongInt)`

Visibility: protected

Description: Writes a Longint of data to a port

See also: `TPortL.readport` ([867](#))

tportl.readport

Synopsis: Reads a Longint of data from a port

Declaration: `function readport(p: LongInt) : LongInt`

Visibility: `protected`

Description: Reads a Longint of data from a port

See also: `TPortL.writeport` ([866](#))

tportl.pp

Synopsis: Access Longint-sized port by port number

Declaration: `Property pp[w: LongInt]: LongInt; default`

Visibility: `public`

Access: `Read,Write`

Description: Access Longint-sized port by port number

23.5 tportw**Description**

The `TPortW` type is implemented specially for access to the ports in a TP compatible manner. There is no need to create an instance of this type: the standard TP variables are instantiated at unit initialization.

Method overview

Page	Method	Description
868	<code>readport</code>	Reads a word of data from a port
867	<code>writeport</code>	Writes a word of data to a port

Property overview

Page	Property	Access	Description
868	<code>pp</code>	<code>rw</code>	Access word-sized port by port number

tportw.writeport

Synopsis: Writes a word of data to a port

Declaration: `procedure writeport(p: LongInt; data: Word)`

Visibility: `protected`

Description: Writes a word of data to a port

See also: `TPortW.readport` ([868](#))

tportw.readport

Synopsis: Reads a word of data from a port

Declaration: `function readport(p: LongInt) : Word`

Visibility: `protected`

Description: Reads a word of data from a port

See also: `TPortW.writeport` ([867](#))

tportw.pp

Synopsis: Access word-sized port by port number

Declaration: `Property pp[w: LongInt]: Word; default`

Visibility: `public`

Access: `Read,Write`

Description: Access word-sized port by port number

Chapter 24

Reference for unit 'printer'

24.1 Overview

This chapter describes the PRINTER unit for Free Pascal. It was written for dos by Florian Klaempfl, and it was written for linux by Michael Van Canneyt, and has been ported to Windows and os/2 as well. Its basic functionality is the same for all supported systems, although there are minor differences on linux/unix.

24.2 Constants, types and variables

Variables

`Lst : text`

`Lst` is the standard printing device.

On linux, `Lst` is set up using `AssignLst('/tmp/PID.lst')`.

24.3 Procedures and functions

AssignLst

Synopsis: Assign text file to printing device

Declaration: `procedure AssignLst(var F: text;ToFile: String)`

Visibility: default

Description: `AssignLst` Assigns to `F` a printing device - *Unix only*. `ToFile` is a string with the following form:

- `'|filename options'` : This sets up a pipe with the program filename, with the given options, such as in the `popen()` call.
- `'filename'` : Prints to file filename. Filename can contain the string 'PID' (No Quotes), which will be replaced by the PID of your program. When closing `lst`, the file will be sent to `lpr` and deleted. (`lpr` should be in `PATH`)
- `{ 'filename| ' }` Idem as previous, only the file is NOT sent to `lpr`, nor is it deleted. (useful for opening `/dev/printer` or for later printing)

Errors: Errors are reported in `Linuxerror`.

See also: `lst` ([869](#))

Listing: `./printex/printex.pp`

```

program testprn;

uses printer;

var i : integer;
    f : text;

begin
  writeln ('Test of printer unit');
  writeln ('Writing to lst...');
  for i:=1 to 80 do writeln (lst, 'This is line ', i, '.' #13);
  close (lst);
  writeln ('Done. ');
  {$ifdef Unix}
  writeln ('Writing to pipe...');
  assignlst (f, '|/usr/bin/lpr -m');
  rewrite (f);
  for i:=1 to 80 do writeln (f, 'This is line ', i, '.' #13);
  close (f);
  writeln ('Done. ')
  {$endif}
end.

```

InitPrinter

Declaration: `procedure InitPrinter(const PrinterName: String)`

Visibility: default

IsLstAvailable

Declaration: `function IsLstAvailable : Boolean`

Visibility: default

Chapter 25

Reference for unit 'Sockets'

25.1 Used units

Table 25.1: Used units by unit 'Sockets'

Name	Page
UnixType	871

25.2 Overview

This document describes the SOCKETS unit for Free Pascal. it was written for linux by Michael Van Canneyt, and ported to Windows by Florian Klaempfl.

25.3 Constants, types and variables

Constants

AF_APPLETALK = 16

Socket domain: Appletalk DDP

AF_ATM = 30

Address family: ?

AF_CCITT = 10

Address family: ?

AF_CHAOS = 5

Address family: ?

AF_CNT = 21

Address family: ?

AF_COIP = 20

Address family: ?

AF_DATAKIT = 9

Address family: ?

AF_DECnet = 12

Address family: Reserved for DECnet project.

AF_DLI = 13

Address family: ?

AF_E164 = AF_ISDN

Address family: ?

AF_ECMA = 8

Address family: ?

AF_FILE = PF_FILE

Address family: Unix socket (alias)

AF_HYLINK = 15

Address family: ?

AF_IMPLINK = 3

Address family: ?

AF_INET = PF_INET

Socket domain: Internet IP Protocol

AF_INET6 = 28

Socket domain: IP version 6

AF_IPX = 23

Socket domain: Novell IPX

AF_ISDN = 26

Address family: ?

AF_ISO = 7

Address family: ?

AF_LAT = 14

Address family: ?

AF_LINK = 18

Address family: ?

AF_LOCAL = PF_LOCAL

Address family: Unix socket

AF_MAX = 33

Socket domain: Maximum value

AF_NATM = 29

Address family: ?

AF_NETGRAPH = 32

Address family: ?

AF_NS = 6

Address family: ?

AF_OSI = AF_ISO

Address family: ?

AF_PUP = 4

Address family: ?

AF_ROUTE = 17

Address family: Alias to emulate 4.4BSD.

AF_SIP = 24

Address family: ?

AF_SNA = 11

Address family: Linux SNA project

AF_UNIX = PF_UNIX

Socket domain: Unix domain sockets

AF_UNSPEC = PF_UNSPEC

Socket domain: Not specified

MSG_CONFIRM = \$0800

Send flags: Conform connection

MSG_CTRUNC = \$0008

Receive flags: Control Data was discarded (buffer too small)

MSG_DONTROUTE = \$0004

Send flags: don't use gateway

MSG_DONTWAIT = \$0040

Receive flags: Non-blocking operation request.

MSG_EOR = \$0080

Receive flags: End of record

MSG_ERRQUEUE = \$2000

Receive flags: ?

MSG_FIN = \$0200

Receive flags: ?

MSG_MORE = \$8000

Receive flags: ?

MSG_NOSIGNAL = \$4000

Receive flags: Suppress SIG_PIPE signal.

MSG_OOB = \$0001

Receive flags: receive out-of-band data.

MSG_PEEK = \$0002

Receive flags: peek at data, don't remove from buffer.

MSG_PROXY = \$0010

Receive flags: ?

MSG_RST = \$1000

Receive flags: ?

MSG_SYN = \$0400

Receive flags: ?

MSG_TRUNC = 0020

Receive flags: packet Data was discarded (buffer too small)

MSG_TRYHARD = MSG_DONTROUTE

Receive flags: ?

MSG_WAITALL = 0100

Receive flags: Wait till operation completed.

PF_APPLETALK = AF_APPLETALK

Protocol family: Appletalk DDP

PF_ATM = AF_ATM

Protocol Family: ?

PF_CCITT = AF_CCITT

Protocol Family: ?

PF_CHAOS = AF_CHAOS

Protocol Family: ?

PF_CNT = AF_CNT

Protocol Family: ?

PF_COIP = AF_COIP

Protocol Family: ?

PF_DATAKIT = AF_DATAKIT

Protocol Family: ?

PF_DECnet = AF_DECnet

Protocol Family: DECNET project

PF_DLI = AF_DLI

Protocol Family: ?

PF_ECMA = AF_ECMA

Protocol Family: ?

PF_FILE = PF_LOCAL

Protocol family: Unix socket (alias)

PF_HYLINK = AF_HYLINK

Protocol Family: ?

PF_IMPLINK = AF_IMPLINK

Protocol Family: ?

PF_INET = 2

Protocol family: Internet IP Protocol

PF_INET6 = AF_INET6

Protocol family: IP version 6

PF_IPX = AF_IPX

Protocol family: Novell IPX

PF_ISDN = AF_ISDN

Protocol Family: ?

PF_ISO = AF_ISO

Protocol Family: ?

PF_KEY = pseudo_AF_KEY

Protocol family: Key management API

PF_LAT = AF_LAT

Protocol Family: ?

PF_LINK = AF_LINK

Protocol Family: ?

PF_LOCAL = 1

Protocol family: Unix socket

PF_MAX = AF_MAX

Protocol family: Maximum value

PF_NATM = AF_NATM

Protocol Family: ?

PF_NETGRAPH = AF_NETGRAPH

Protocol Family: ?

PF_NS = AF_NS

Protocol Family: ?

PF_OSI = AF_ISO

Protocol Family: ?

PF_PIP = pseudo_AF_PIP

Protocol Family: ?

PF_PUP = AF_PUP

Protocol Family: ?

PF_ROUTE = AF_ROUTE

Protocol Family: ?

PF_RTIP = pseudo_AF_RTIP

Protocol Family: ?

PF_SIP = AF_SIP

Protocol Family: ?

PF_SNA = AF_SNA

Protocol Family: Linux SNA project

PF_UNIX = PF_LOCAL

Protocol family: Unix domain sockets

PF_UNSPEC = 0

Protocol family: Unspecified

PF_XTP = pseudo_AF_XTP

Protocol Family: ?

pseudo_AF_HDRCMPLT = 31

Address family: ?

pseudo_AF_KEY = 27

Address family: key management API.

pseudo_AF_PIP = 25

Address family: ?

`pseudo_AF_RTIP` = 22

Address family: ?

`pseudo_AF_XTP` = 19

Address family: ?

`SCM_TIMESTAMP` = `SO_TIMESTAMP`

Socket option: ?

`SHUT_RD` = 0

Shutdown read part of full duplex socket

`SHUT_RDWR` = 2

Shutdown read and write part of full duplex socket

`SHUT_WR` = 1

Shutdown write part of full duplex socket

`SOCK_DGRAM` = 2

Type of socket: datagram (conn.less) socket (UDP)

`SOCK_MAXADDRLLEN` = 255

Maximum socket address length for Bind (884) call.

`SOCK_RAW` = 3

Type of socket: raw socket

`SOCK_RDM` = 4

Type of socket: reliably-delivered message

`SOCK_SEQPACKET` = 5

Type of socket: sequential packet socket

`SOCK_STREAM` = 1

Type of socket: stream (connection) type socket (TCP)

`SOL_SOCKET` = 1

Socket option level: Socket level

`SO_ACCEPTCONN` = 30

Socket option: ?

SO_ATTACH_FILTER = 26

Socket option: ?

SO_BINDTODEVICE = 25

Socket option: ?

SO_BROADCAST = 6

Socket option: Broadcast

SO_BSDCOMPAT = 14

Socket option: ?

SO_DEBUG = 1

Socket option level: debug

SO_DETACH_FILTER = 27

Socket option: ?

SO_DONTROUTE = 5

Socket option: Don't route

SO_ERROR = 4

Socket option: Error

SO_KEEPALIVE = 9

Socket option: keep alive

SO_LINGER = 13

Socket option: ?

SO_NO_CHECK = 11

Socket option: ?

SO_OOBINLINE = 10

Socket option: ?

SO_PASSCRED = 16

Socket option: ?

SO_PEERCREC = 17

Socket option: ?

SO_PEERNAME = 28

Socket option: ?

SO_PRIORITY = 12

Socket option: ?

SO_RCVBUF = 8

Socket option: receive buffer

SO_RCVLOWAT = 18

Socket option: ?

SO_RCVTIMEO = 20

Socket option: ?

SO_REUSEADDR = 2

Socket option: Reuse address

SO_SECURITY_AUTHENTICATION = 22

Socket option: ?

SO_SECURITY_ENCRYPTION_NETWORK = 24

Socket option: ?

SO_SECURITY_ENCRYPTION_TRANSPORT = 23

Socket option: ?

SO_SNDBUF = 7

Socket option: Send buffer

SO_SNDLOWAT = 19

Socket option: ?

SO_SNDTIMEO = 21

Socket option: ?

SO_TIMESTAMP = 29

Socket option: ?

SO_TYPE = 3

Socket option: Type

`S_IN = 0`

Input socket in socket pair.

`S_OUT = 1`

Output socket in socket pair

Types

```
in_addr = packed record
  s_addr : cuint32;
end
```

General inet socket address.

`pIn6_Addr = ^Tin6_addr`

Pointer to `Tin6_addr` ([881](#))

`pInetSockAddr = ^TInetSockAddr`

Pointer to `TInetSockAddr` ([881](#))

`pSockAddr = ^TSockAddr`

Pointer to `TSockAddr` ([882](#))

`psockaddr_in6 = ^sockaddr_in6`

Pointer to `sockaddr_in6` ([881](#))

`sa_family_t = cushort`

Address family type

`sockaddr_in6 = TInetSockAddr6`

Alias for `TInetSockAddr6` ([882](#))

```
Tin6_addr = packed record
end
```

Alias for `TInetSockAddr6` ([882](#))

```
TInetSockAddr = packed record
end
```

`TUnixSockAddr` is used to store a INET socket address for the `Bind` ([884](#)), `Recv` ([891](#)) and `Send` ([892](#)) calls.

```

TInetSockAddr6 = packed record
  sin6_family : sa_family_t;
  sin6_port : cuint16;
  sin6_flowinfo : cuint32;
  sin6_addr : Tin6_addr;
  sin6_scope_id : cuint32;
end

```

Record for IPV6 socket address.

```

TSockAddr = packed record
  sa_family : sa_family_t;
  sa_data : Array[0..13] of Char;
end

```

TUnixSockAddr is used to store a general socket address for the Bind (884), Recv (891) and Send (892) calls.

```

TSockArray = Array[1..2] of LongInt

```

Type returned by the SocketPair (895) call.

```

TSockPairArray = Array[0..1] of LongInt

```

Array of sockets, used in SocketPair (895) call.

```

TUnixSockAddr = packed record
  family : sa_family_t;
  path : Array[0..107] of Char;
end

```

TUnixSockAddr is used to store a UNIX socket address for the Bind (884), Recv (891) and Send (892) calls.

Variables

```

SocketError : cint

```

SocketError contains the error code for the last socket operation. It can be examined to return the last socket error.

25.4 Procedures and functions

Accept

Synopsis: Accept a connection from a socket.

Declaration: function Accept(Sock: LongInt; var Addr; var Addrlen: LongInt) : LongInt
 function Accept(Sock: LongInt; var addr: TInetSockAddr; var SockIn: File;
 var SockOut: File) : Boolean

```

function Accept(Sock: LongInt;var addr: TInetSockAddr;var SockIn: text;
               var SockOut: text) : Boolean
function Accept(Sock: LongInt;var addr: String;var SockIn: text;
               var SockOut: text) : Boolean
function Accept(Sock: LongInt;var addr: String;var SockIn: File;
               var SockOut: File) : Boolean

```

Visibility: default

Description: FPAccept accepts a connection from a socket Sock, which was listening for a connection. If a connection is accepted, a file descriptor is returned. On error -1 is returned. The returned socket may NOT be used to accept more connections. The original socket remains open.

The Accept call fills the address of the connecting entity in Addr, and sets its length in AddrLen. Addr should be pointing to enough space, and AddrLen should be set to the amount of space available, prior to the call.

The alternate forms of the Accept (882) command, with the Text or File parameters are equivalent to subsequently calling the regular Accept (882) function and the Sock2Text (894) or Sock2File (894) functions. These functions return True if successful, False otherwise.

Errors: On error, -1 is returned, and errors are reported in SocketError, and include the following:

SYS_EBADFThe socket descriptor is invalid.

SYS_ENOTSOCKThe descriptor is not a socket.

SYS_EOPNOTSUPPThe socket type doesn't support the Listen operation.

SYS_EFAULTAddr points outside your address space.

SYS_EWOULDBLOCKThe requested operation would block the process.

See also: Listen (891), Connect (885), Bind (884)

Listing: ./sockex/socksvr.pp

Program server;

```

{
  Program to test Sockets unit by Michael van Canneyt and Peter Vreman
  Server Version, First Run sock_svr to let it create a socket and then
  sock_cli to connect to that socket
}

```

uses BaseUnix, Sockets;

const

 SPath='ServerSoc';

Var

 FromName : **string**;

 Buffer : **string**[255];

 S : Longint;

 Sin, Sout : Text;

procedure perror (**const** S:**string**);

begin

 writeln (S, SocketError);

 halt(100);

end;


```

begin
  S:=Socket (AF_UNIX,SOCK_STREAM,0);
  if SocketError<>0 then
    Perror ( 'Server : Socket : ');
  fpUnLink(SPath);
  if not Bind(S,SPath) then
    Perror ( 'Server : Bind : ');
  if not Listen (S,1) then
    Perror ( 'Server : Listen : ');
  WriteLn('Waiting for Connect from Client , run now sock_cli in an other tty');
  if not Accept (S,FromName,Sin,Sout) then
    Perror ( 'Server : Accept : '+fromname);
  Reset(Sin);
  ReWrite(Sout);
  WriteLn(Sout,'Message From Server');
  Flush(SOut);
  while not eof(sin) do
    begin
      ReadLn(Sin,Buffer);
      WriteLn('Server : read : ',buffer);
    end;
  FPUntLink(SPath);
end.

```

Bind

Synopsis: Bind a socket to an address.

Declaration: function Bind(Sock: LongInt;const Addr;AddrLen: LongInt) : Boolean
 function Bind(Sock: LongInt;const addr: String) : Boolean

Visibility: default

Description: Bind binds the socket Sock to address Addr. Addr has length AddrLen. The function returns True if the call was succesful, False if not.

The form of the Bind command with the TUnixSockAddr (882) is equivalent to subsequently calling Str2UnixSockAddr (895) and the regular Bind function. The function returns True if successfull, False otherwise.

Errors: Errors are returned in SocketError and include the following:

SYS_EBADFThe socket descriptor is invalid.

SYS_EINVALThe socket is already bound to an address,

SYS_EACCESSAddress is protected and you don't have permission to open it.

More errors can be found in the Unix man pages.

See also: Socket (895)

CloseSocket

Synopsis: Closes a socket handle.

Declaration: function CloseSocket(Sock: LongInt) : LongInt

Visibility: default

Description: `CloseSocket` closes a socket handle. It returns 0 if the socket was closed successfully, -1 if it failed.

Errors: On error, -1 is returned.

See also: `Socket` ([895](#))

Connect

Synopsis: Open a connection to a server socket.

Declaration: `function Connect(Sock: LongInt;const Addr;AddrLen: LongInt) : Boolean`
`function Connect(Sock: LongInt;const addr: TInetSockAddr;`
`var SockIn: text;var SockOut: text) : Boolean`
`function Connect(Sock: LongInt;const addr: TInetSockAddr;`
`var SockIn: file;var SockOut: file) : Boolean`
`function Connect(Sock: LongInt;const addr: String;var SockIn: text;`
`var SockOut: text) : Boolean`
`function Connect(Sock: LongInt;const addr: String;var SockIn: file;`
`var SockOut: file) : Boolean`

Visibility: default

Description: `Connect` opens a connection to a peer, whose address is described by `Addr`. `AddrLen` contains the length of the address. The type of `Addr` depends on the kind of connection you're trying to make, but is generally one of `TSockAddr` or `TUnixSockAddr`.

The forms of the `Connect` ([885](#)) command with the `Text` or `File` arguments are equivalent to subsequently calling the regular `Connect` function and the `Sock2Text` ([894](#)) or `Sock2File` ([894](#)) functions. These functions return `True` if successful, `False` otherwise.

The `Connect` function returns a file descriptor if the call was successful, -1 in case of error.

Errors: On error, -1 is returned and errors are reported in `SocketError`.

See also: `Listen` ([891](#)), `Bind` ([884](#)), `Accept` ([882](#))

Listing: `./sockex/sockcli.pp`

Program `Client`;

```
{
  Program to test Sockets unit by Michael van Canneyt and Peter Vreman
  Client Version, First Run sock_svr to let it create a socket and then
  sock_cli to connect to that socket
}
```

uses `Sockets, BaseUnix`;

```
procedure PError(const S : string);
begin
  writeln(S, SocketError);
  halt(100);
end;
```

Var

```

Saddr    : String [25];
Buffer   : string [255];
S        : Longint;
Sin, Sout : Text;
i        : integer;
begin
  S:=Socket (AF_UNIX,SOCK_STREAM,0);
  if SocketError<>0 then
    PError('Client : Socket : ');
  Saddr:= 'ServerSoc';
  if not Connect (S,SAddr,Sin,Sout) then
    PError('Client : Connect : ');
  Reset(Sin);
  ReWrite(Sout);
  Buffer:= 'This is a textstring sent by the Client.';
  for i:=1 to 10 do
    WriteLn(Sout, Buffer);
  Flush(Sout);
  ReadLn(Sin, Buffer);
  WriteLn(Buffer);
  Close(sout);
end.

```

Listing: ./sockex/pfinger.pp

```

program pfinger;

uses sockets, errors;

Var Addr : TInetSockAddr;
    S : Longint;
    Sin, Sout : Text;
    Line : string;

begin
  Addr.family:=AF_INET;
  { port 79 in network order }
  Addr.port:=79 shl 8;
  { localhost : 127.0.0.1 in network order }
  Addr.addr:=((1 shl 24) or 127);
  S:=Socket(AF_INET,SOCK_STREAM,0);
  If Not Connect (S,Addr,SIN,SOUT) Then
    begin
      WriteLn ( 'Couldn't connect to localhost');
      WriteLn ( 'Socket error : ',strerror(SocketError));
      halt(1);
    end;
  rewrite (sout);
  reset(sin);
  writeln (sout, paramstr(1));
  flush(sout);
  while not eof(sin) do
    begin
      readLn (Sin, line);
      writeln (line);
    end;
  Shutdown(s,2);
  close (sin);
  close (sout);

```

end.

fpaccept

Synopsis: Alias for the `accept` (882) call

Declaration: `function fpaccept(s: cint;addrx: pSockAddr;addrlen: psocklen) : cint`

Visibility: default

fpbind

Synopsis: Alias for the `bind` (884) call

Declaration: `function fpbind(s: cint;addrx: pSockAddr;addrlen: tsocklen) : cint`

Visibility: default

fpconnect

Synopsis: Alias for the `connect` (885) call

Declaration: `function fpconnect(s: cint;name: pSockAddr;namelen: tsocklen) : cint`

Visibility: default

fpgetpeername

Synopsis: Alias for the `GetPeerName` (889) call

Declaration: `function fpgetpeername(s: cint;name: pSockAddr;namelen: psocklen) : cint`

Visibility: default

fpgetsockname

Synopsis: Alias for the `GetSocketName` (889) call

Declaration: `function fpgetsockname(s: cint;name: pSockAddr;namelen: psocklen) : cint`

Visibility: default

fpgetsockopt

Synopsis: Alias for the `GetSocketOptions` (890) call

Declaration: `function fpgetsockopt(s: cint;level: cint;optname: cint;optval: pointer;
optlen: psocklen) : cint`

Visibility: default

fplisten

Synopsis: Alias for the listen ([891](#)) call

Declaration: `function fplisten(s: cint;backlog: cint) : cint`

Visibility: default

fprecv

Synopsis: Alias for the recv ([891](#)) call

Declaration: `function fprecv(s: cint;buf: pointer;len: size_t;flags: cint) : ssize_t`

Visibility: default

fprecvfrom

Synopsis: Alias for the recvfrom ([892](#)) call

Declaration: `function fprecvfrom(s: cint;buf: pointer;len: size_t;flags: cint;
from: pSockAddr;fromlen: psocklen) : ssize_t`

Visibility: default

fpsend

Synopsis: Alias for the send ([892](#)) call

Declaration: `function fpsend(s: cint;msg: pointer;len: size_t;flags: cint) : ssize_t`

Visibility: default

fpsendto

Synopsis: Alias for the send ([892](#)) call

Declaration: `function fpsendto(s: cint;msg: pointer;len: size_t;flags: cint;
tox: pSockAddr;tolen: tsocklen) : ssize_t`

Visibility: default

fpsetsockopt

Synopsis: Alias for the SetSocketOptions ([893](#)) call

Declaration: `function fpsetsockopt(s: cint;level: cint;optname: cint;optval: pointer;
optlen: tsocklen) : cint`

Visibility: default

fpshutdown

Synopsis: Alias for the shutdown ([894](#)) call

Declaration: `function fpshutdown(s: cint;how: cint) : cint`

Visibility: default

fpsocket

Synopsis: Alias for the socket (895) call

Declaration: `function fpsocket(domain: cint; xtype: cint; protocol: cint) : cint`

Visibility: default

fpsocketpair

Synopsis: Alias for the SocketPair (895) call

Declaration: `function fpsocketpair(d: cint; xtype: cint; protocol: cint; sv: pcint) : cint`

Visibility: default

GetPeerName

Synopsis: Return the name (address) of the connected peer.

Declaration: `function GetPeerName(Sock: LongInt; var Addr; var AddrLen: LongInt) : LongInt`

Visibility: default

Description: `GetPeerName` returns the name of the entity connected to the specified socket `Sock`. The Socket must be connected for this call to work.

`Addr` should point to enough space to store the name, the amount of space pointed to should be set in `AddrLen`. When the function returns successfully, `Addr` will be filled with the name, and `AddrLen` will be set to the length of `Addr`.

Errors: Errors are reported in `SocketError`, and include the following:

SYS_EBADFThe socket descriptor is invalid.

SYS_ENOBUFSThe system doesn't have enough buffers to perform the operation.

SYS_ENOTSOCKThe descriptor is not a socket.

SYS_EFAULT`Addr` points outside your address space.

SYS_ENOTCONNThe socket isn't connected.

See also: `Connect` (885), `Socket` (895)

GetSocketName

Synopsis: Return name of socket.

Declaration: `function GetSocketName(Sock: LongInt; var Addr; var AddrLen: LongInt) : LongInt`

Visibility: default

Description: `GetSockName` returns the current name of the specified socket `Sock`. `Addr` should point to enough space to store the name, the amount of space pointed to should be set in `AddrLen`. When the function returns successfully, `Addr` will be filled with the name, and `AddrLen` will be set to the length of `Addr`.

Errors: Errors are reported in `SocketError`, and include the following:

SYS_EBADFThe socket descriptor is invalid.

SYS_ENOBUFSThe system doesn't have enough buffers to perform the operation.

SYS_ENOTSOCKThe descriptor is not a socket.

SYS_EFAULT`Addr` points outside your address space.

See also: `Bind` ([884](#))

GetSocketOptions

Synopsis: Get current socket options

Declaration: `function GetSocketOptions(Sock: LongInt; Level: LongInt; OptName: LongInt;
var OptVal; var optlen: LongInt) : LongInt`

Visibility: default

Description: `GetSocketOptions` gets the connection options for socket `Sock`. The socket may be obtained from different levels, indicated by `Level`, which can be one of the following:

SOL_SOCKETFrom the socket itself.

XXXset `Level` to `XXX`, the protocol number of the protocol which should interpret the option.

For more information on this call, refer to the unix manual page `\seem{getsockopt}{2}`.

Errors: Errors are reported in `SocketError`, and include the following:

SYS_EBADFThe socket descriptor is invalid.

SYS_ENOTSOCKThe descriptor is not a socket.

SYS_EFAULT`OptVal` points outside your address space.

See also: `GetSocketOptions` ([890](#))

htonl

Synopsis: Convert long integer from host ordered to network ordered

Declaration: `function htonl(host: LongInt) : LongInt`

Visibility: default

Description: `htonl` makes sure that the bytes in `host` are ordered in the correct way for sending over the network and returns the correctly ordered result.

See also: `htons` ([890](#)), `ntohl` ([891](#)), `ntohs` ([891](#))

htons

Synopsis: Convert short integer from host ordered to network ordered

Declaration: `function htons(host: Word) : Word`

Visibility: default

Description: `htons` makes sure that the bytes in `host` are ordered in the correct way for sending over the network and returns the correctly ordered result.

See also: `htonl` ([890](#)), `ntohl` ([891](#)), `ntohs` ([891](#))

Listen

Synopsis: Listen for connections on socket.

Declaration: `function Listen(Sock: LongInt;MaxConnect: LongInt) : Boolean`

Visibility: default

Description: `Listen` listens for up to `MaxConnect` connections from socket `Sock`. The socket `Sock` must be of type `SOCK_STREAM` or `Sock_SEQPACKET`.

The function returns `True` if a connection was accepted, `False` if an error occurred.

Errors: Errors are reported in `SocketError`, and include the following:

SYS_EBADFThe socket descriptor is invalid.

SYS_ENOTSOCKThe descriptor is not a socket.

SYS_EOPNOTSUPPThe socket type doesn't support the `Listen` operation.

See also: `Socket` ([895](#)), `Bind` ([884](#)), `Connect` ([885](#))

NToHl

Synopsis: Convert long integer from network ordered to host ordered

Declaration: `function NToHl(Net: LongInt) : LongInt`

Visibility: default

Description: `ntohs` makes sure that the bytes in `Net`, received from the network, are ordered in the correct way for handling by the host machine, and returns the correctly ordered result.

See also: `htonl` ([890](#)), `htons` ([890](#)), `ntohs` ([891](#))

NToHs

Synopsis: Convert short integer from network ordered to host ordered

Declaration: `function NToHs(Net: Word) : Word`

Visibility: default

Description: `ntohs` makes sure that the bytes in `Net`, received from the network, are ordered in the correct way for handling by the host machine, and returns the correctly ordered result.

See also: `htonl` ([890](#)), `htons` ([890](#)), `ntohl` ([891](#))

Recv

Synopsis: Receive data on socket

Declaration: `function Recv(Sock: LongInt;var Buf;BufLen: LongInt;Flags: LongInt)
: LongInt`

Visibility: default

Description: `Recv` reads at most `AddrLen` bytes from socket `Sock` into address `Addr`. The socket must be in a connected state. `Flags` can be one of the following:

1Process out-of band data.

4Bypass routing, use a direct interface.

??Wait for full request or report an error.

The functions returns the number of bytes actually read from the socket, or -1 if a detectable error occurred.

Errors: Errors are reported in `SocketError`, and include the following:

SYS_EBADFThe socket descriptor is invalid.

SYS_ENOTCONNThe socket isn't connected.

SYS_ENOTSOCKThe descriptor is not a socket.

SYS_EFAULTThe address is outside your address space.

SYS_EMSGSIZEThe message cannot be sent atomically.

SYS_EWOULDBLOCKThe requested operation would block the process.

SYS_ENOBUFSThe system doesn't have enough free buffers available.

See also: `Send` ([892](#))

RecvFrom

Synopsis: Receive data from an unconnected socket

Declaration: `function RecvFrom(Sock: LongInt; var Buf; Buflen: LongInt; Flags: LongInt; var Addr; var AddrLen: LongInt) : LongInt`

Visibility: default

Description: `RecvFrom` receives data in buffer `Buf` with maximum length `Buflen` from socket `Sock`. Receipt is controlled by options in `Flags`. `Addr` will be filled with the address from the sender, and will have length `AddrLen`. The function returns the number of bytes received, or -1 on error. `AddrLen`.

Errors: On error, -1 is returned.

See also: `Socket` ([895](#)), `recv` ([891](#)), `RecvFrom` ([892](#))

Send

Synopsis: Send data through socket

Declaration: `function Send(Sock: LongInt; const Buf; Buflen: LongInt; Flags: LongInt) : LongInt`

Visibility: default

Description: `Send` sends `AddrLen` bytes starting from address `Addr` to socket `Sock`. `Sock` must be in a connected state. The function returns the number of bytes sent, or -1 if a detectable error occurred.

`Flags` can be one of the following:

1Process out-of band data.

4Bypass routing, use a direct interface.

Errors: Errors are reported in `SocketError`, and include the following:

SYS_EBADFThe socket descriptor is invalid.

SYS_ENOTSOCKThe descriptor is not a socket.

SYS_EFAULTThe address is outside your address space.

SYS_EMSGSIZEThe message cannot be sent atomically.

SYS_EWOULDBLOCKThe requested operation would block the process.

SYS_ENOBUFSThe system doesn't have enough free buffers available.

See also: [Recv \(891\)](#)

SendTo

Synopsis: Send data through an unconnected socket to an address.

Declaration:

```
function SendTo(Sock: LongInt; const Buf: BufLen: LongInt; Flags: LongInt;
                var Addr: AddrLen: LongInt) : LongInt
```

Visibility: default

Description: `SendTo` sends data from buffer `Buf` with length `Buflen` through socket `Sock` with options `Flags`. The data is sent to address `Addr`, which has length `AddrLen`

Errors: On error, -1 is returned.

See also: [Socket \(895\)](#), [Send \(892\)](#), [RecvFrom \(892\)](#)

SetSocketOptions

Synopsis: Set socket options.

Declaration:

```
function SetSocketOptions(Sock: LongInt; Level: LongInt; OptName: LongInt;
                          const OptVal; optlen: LongInt) : LongInt
```

Visibility: default

Description: `SetSocketOptions` sets the connection options for socket `Sock`. The socket may be manipulated at different levels, indicated by `Level`, which can be one of the following:

SOL_SOCKETTo manipulate the socket itself.

XXXset `Level` to `XXX`, the protocol number of the protocol which should interpret the option.

For more information on this call, refer to the unix manual page `setsockopt`

Errors: Errors are reported in `SocketError`, and include the following:

SYS_EBADFThe socket descriptor is invalid.

SYS_ENOTSOCKThe descriptor is not a socket.

SYS_EFAULT`OptVal` points outside your address space.

See also: [GetSocketOptions \(890\)](#)

Shutdown

Synopsis: Close one end of full duplex connection.

Declaration: `function Shutdown(Sock: LongInt;How: LongInt) : LongInt`

Visibility: default

Description: `ShutDown` closes one end of a full duplex socket connection, described by `Sock`. The parameter `How` determines how the connection will be shut down, and can be one of the following:

0Further receives are disallowed.

1Further sends are disallowed.

2Sending nor receiving are allowed.

On succes, the function returns 0, on error -1 is returned.

Errors: `SocketError` is used to report errors, and includes the following:

SYS_EBADFThe socket descriptor is invalid.

SYS_ENOTCONNThe socket isn't connected.

SYS_ENOTSOCKThe descriptor is not a socket.

See also: `Socket` ([895](#)), `Connect` ([885](#))

Sock2File

Synopsis: Convert socket to untyped file descriptors

Declaration: `procedure Sock2File(Sock: LongInt;var SockIn: File;var SockOut: File)`

Visibility: default

Description: `Sock2File` transforms a socket `Sock` into 2 Pascal file descriptors of type `File`, one for reading from the socket (`SockIn`), one for writing to the socket (`SockOut`).

Errors: None.

See also: `Socket` ([895](#)), `Sock2Text` ([894](#))

Sock2Text

Synopsis: Convert socket to text file descriptors

Declaration: `procedure Sock2Text(Sock: LongInt;var SockIn: Text;var SockOut: Text)`

Visibility: default

Description: `Sock2Text` transforms a socket `Sock` into 2 Pascal file descriptors of type `Text`, one for reading from the socket (`SockIn`), one for writing to the socket (`SockOut`).

Errors: None.

See also: `Socket` ([895](#)), `Sock2File` ([894](#))

Socket

Synopsis: Create new socket

Declaration: `function Socket(Domain: LongInt; SocketType: LongInt; Protocol: LongInt)
: LongInt`

Visibility: default

Description: `Socket` creates a new socket in domain `Domain`, from type `SocketType` using protocol `Protocol`.

The `Domain`, `Socket` type and `Protocol` can be specified using predefined constants (see the section on constants for available constants) If successful, the function returns a socket descriptor, which can be passed to a subsequent `Bind` (884) call. If unsuccessful, the function returns -1.

for an example, see `Accept` (882).

Errors: Errors are returned in `SocketError`, and include the following:

SYS_EPROTONOSUPPORT The protocol type or the specified protocol is not supported within this domain.

SYS_EMFILE The per-process descriptor table is full.

SYS_ENFILE The system file table is full.

SYS_EACCESS Permission to create a socket of the specified type and/or protocol is denied.

SYS_ENOBUFS Insufficient buffer space is available. The socket cannot be created until sufficient resources are freed.

See also: `SocketPair` (895)

SocketPair

Synopsis: Create socket pair

Declaration: `function SocketPair(Domain: LongInt; SocketType: LongInt;
Protocol: LongInt; var Pair: TSocketArray) : LongInt`

Visibility: default

Description: `SocketPair` creates 2 sockets in domain `Domain`, from type `SocketType` and using protocol `Protocol`. The pair is returned in `Pair`, and they are indistinguishable. The function returns -1 upon error and 0 upon success.

Errors: Errors are reported in `SocketError`, and are the same as in `Socket` (895)

See also: `Str2UnixSockAddr` (895)

Str2UnixSockAddr

Synopsis: Convert path to `TUnixSockAddr` (882)

Declaration: `procedure Str2UnixSockAddr(const addr: String; var t: TUnixSockAddr;
var len: LongInt)`

Visibility: default

Description: `Str2UnixSockAddr` transforms a Unix socket address in a string to a `TUnixSockAddr` structure which can be passed to the `Bind` (884) call.

Errors: None.

See also: `Socket` (895), `Bind` (884)

Chapter 26

Reference for unit 'strings'

26.1 Overview

This chapter describes the `STRINGS` unit for Free Pascal. This unit is system independent, and therefore works on all supported platforms.

26.2 Procedures and functions

stralloc

Synopsis: Allocate memory for a new null-terminated string on the heap

Declaration: `function stralloc(L: SizeInt) : pchar`

Visibility: default

Description: `StrAlloc` reserves memory on the heap for a string with length `Len`, terminating `#0` included, and returns a pointer to it.

Errors: If there is not enough memory, a run-time error occurs.

See also: `StrNew` ([904](#)), `StrPCopy` ([905](#))

strcat

Synopsis: Concatenate 2 null-terminated strings.

Declaration: `function strcat(dest: pchar; source: pchar) : pchar`

Visibility: default

Description: Attaches `Source` to `Dest` and returns `Dest`.

Errors: No length checking is performed.

See also: `StrLCat` ([900](#))

Listing: `./stringex/ex11.pp`

```

Program Example11;

Uses strings;

{ Program to demonstrate the StrCat function. }

Const P1 : PChar = 'This is a PChar String.';

Var P2 : PChar;

begin
  P2:= StrAlloc ( StrLen(P1)*2+1);
  StrMove (P2,P1,StrLen(P1)+1); { P2=P1 }
  StrCat (P2,P1);                { Append P2 once more }
  WriteLn ( 'P2 : ',P2);
end.

```

strcmp

Synopsis: Compare 2 null-terminated strings, case sensitive.

Declaration: `function strcmp(str1: pchar;str2: pchar) : SizeInt`

Visibility: default

Description: Compares the null-terminated strings S1 and S2. The result is

- A negative Longint when S1<S2.
- 0 when S1=S2.
- A positive Longint when S1>S2.

For an example, see StrLComp (901).

Errors: None.

See also: StrLComp (901), StrIComp (900), StrLComp (903)

strcpy

Synopsis: Copy a null-terminated string

Declaration: `function strcpy(dest: pchar;source: pchar) : pchar`

Visibility: default

Description: Copy the null terminated string in Source to Dest, and returns a pointer to Dest. Dest needs enough room to contain Source, i.e. `StrLen(Source)+1` bytes.

Errors: No length checking is performed.

See also: StrPCopy (905), StrLCopy (902), StrECopy (899)

Listing: ./stringex/ex4.pp

```

Program Example4;

Uses strings;

{ Program to demonstrate the StrCopy function. }

Const P : PChar = 'This is a PCHAR string.';

var PP : PChar;

begin
  PP:= StrAlloc ( Strlen (P)+1);
  STrCopy (PP,P);
  If StrComp (PP,P)<>0 then
    Writeln ( 'Oh-oh problems...' )
  else
    Writeln ( 'All is well : PP=',PP);
end.

```

strdispose

Synopsis: disposes of a null-terminated string on the heap

Declaration: `procedure strdispose(p: pchar)`

Visibility: default

Description: Removes the string in P from the heap and releases the memory.

Errors: None.

See also: StrNew ([904](#))

Listing: ./stringex/ex17.pp

```

Program Example17;

Uses strings;

{ Program to demonstrate the StrDispose function. }

Const P1 : PChar = 'This is a PChar string';

var P2 : PChar;

begin
  Writeln ( 'Before StrNew : Memory available : ',MemAvail);
  P2:=StrNew (P1);
  Writeln ( 'After StrNew : Memory available : ',MemAvail);
  Writeln ( 'P2 : ',P2);
  StrDispose(P2);
  Writeln ( 'After StrDispose : Memory available : ',MemAvail);
end.

```

strcpy

Synopsis: Copy a null-terminated string, return a pointer to the end.

Declaration: `function strcpy(dest: pchar;source: pchar) : pchar`

Visibility: default

Description: Copies the Null-terminated string in Source to Dest, and returns a pointer to the end (i.e. the terminating Null-character) of the copied string.

Errors: No length checking is performed.

See also: [StrLCopy \(902\)](#), [StrCopy \(897\)](#)

Listing: ./stringex/ex6.pp

Program Example6;

Uses strings;

{ Program to demonstrate the StrECopy function. }

Const P : PChar = 'This is a PCHAR string.';

Var PP : PChar;

begin

PP:=StrAlloc (StrLen(P)+1);

If Longint(StrECopy(PP,P)) – Longint(PP) <> StrLen(P) **then**

Writeln ('Something is wrong here !')

else

Writeln ('PP= ',PP);

end.

strend

Synopsis: Return a pointer to the end of a null-terminated string

Declaration: `function strend(p: pchar) : pchar`

Visibility: default

Description: Returns a pointer to the end of P. (i.e. to the terminating null-character.

Errors: None.

See also: [StrLen \(902\)](#)

Listing: ./stringex/ex7.pp

Program Example6;

Uses strings;

{ Program to demonstrate the StrEnd function. }

Const P : PChar = 'This is a PCHAR string.';

```

begin
  If Longint(StrEnd(P)) - Longint(P) <> StrLen(P) then
    Writeln('Something is wrong here !')
  else
    Writeln('All is well..');
end.

```

stricmp

Synopsis: Compare 2 null-terminated strings, case insensitive.

Declaration: `function stricmp(str1: pchar; str2: pchar) : SizeInt`

Visibility: default

Description: Compares the null-terminated strings S1 and S2, ignoring case. The result is

- A negative Longint when S1 < S2.
- 0 when S1 = S2.
- A positive Longint when S1 > S2.

Errors: None.

See also: StrLComp ([901](#)), StrComp ([897](#)), StrLComp ([903](#))

Listing: ./stringex/ex8.pp

Program Example8;

Uses strings;

{ Program to demonstrate the StrLComp function. }

```

Const P1 : PChar = 'This is the first string.';
      P2 : PChar = 'This is the second string.';

```

```

Var L : Longint;

```

```

begin
  Write('P1 and P2 are ');
  If StrComp(P1, P2) <> 0 then write('NOT ');
  write('equal. The first ');
  L := 1;
  While StrLComp(P1, P2, L) = 0 do inc(L);
  dec(L);
  Writeln(L, ' characters are the same.');
```

strlcat

Synopsis: Concatenate 2 null-terminated strings, with length boundary.

Declaration: `function strlcat(dest: pchar; source: pchar; l: SizeInt) : pchar`

Visibility: default

Description: Adds MaxLen characters from Source to Dest, and adds a terminating null-character. Returns Dest.

Errors: None.

See also: StrCat ([896](#))

Listing: ./stringex/ex12.pp

Program Example12;

Uses strings;

{ Program to demonstrate the StrLCat function. }

Const P1 : PChar = '1234567890';

Var P2 : PChar;

begin

P2:= StrAlloc (**StrLen**(P1)*2+1);

P2^:=#0; *{ Zero length }*

StrCat (P2,P1);

StrLCat (P2,P1,5);

WriteLn ('P2 = ',P2);

end.

strlcomp

Synopsis: Compare limited number of characters of 2 null-terminated strings

Declaration: function strlcomp(str1: pchar;str2: pchar;l: SizeInt) : SizeInt

Visibility: default

Description: Compares maximum L characters of the null-terminated strings S1 and S2. The result is

- A negative Longint when S1<S2.
- 0 when S1=S2.
- A positive Longint when S1>S2.

Errors: None.

See also: StrComp ([897](#)), StrIComp ([900](#)), StrLComp ([903](#))

Listing: ./stringex/ex8.pp

Program Example8;

Uses strings;

{ Program to demonstrate the StrLComp function. }

Const P1 : PChar = 'This is the first string.';

P2 : PChar = 'This is the second string.';

Var L : Longint;

```

begin
  Write ( 'P1 and P2 are ');
  If StrComp (P1,P2)<>0 then write ( 'NOT ');
  write ( 'equal. The first ');
  L:=1;
  While StrLComp(P1,P2,L)=0 do inc (L);
  dec(L);
  Writeln (L, ' characters are the same. ');
end.

```

strlcopy

Synopsis: Copy a null-terminated string, limited in length.

Declaration: `function strlcopy(dest: pchar;source: pchar;maxlen: SizeInt) : pchar`

Visibility: default

Description: Copies MaxLen characters from Source to Dest, and makes Dest a null terminated string.

Errors: No length checking is performed.

See also: StrCopy ([897](#)), StrECopy ([899](#))

Listing: ./stringex/ex5.pp

Program Example5;

Uses strings;

{ Program to demonstrate the StrLCopy function. }

Const P : PChar = '123456789ABCDEF';

var PP : PChar;

begin

PP:=StrAlloc(11);

Writeln ('First 10 characters of P : ',StrLCopy (PP,P,10));

end.

strlen

Synopsis: Length of a null-terminated string.

Declaration: `function strlen(p: pchar) : sizeint`

Visibility: default

Description: Returns the length of the null-terminated string P.

Errors: None.

See also: StrNew ([904](#))

Listing: ./stringex/ex1.pp

```

Program Example1;

Uses strings;

{ Program to demonstrate the StrLen function. }

Const P : PChar = 'This is a constant pchar string';

begin
  WriteLn ( 'P          : ',p);
  WriteLn ( 'length(P) : ',StrLen(P));
end.

```

strlicomp

Synopsis: Compare limited number of characters in 2 null-terminated strings, ignoring case.

Declaration: `function strlicomp(str1: pchar;str2: pchar;l: SizeInt) : SizeInt`

Visibility: default

Description: Compares maximum L characters of the null-terminated strings S1 and S2, ignoring case. The result is

- A negative Longint when S1<S2.
- 0 when S1=S2.
- A positive Longint when S1>S2.

For an example, see StrIComp ([900](#))

Errors: None.

See also: StrLComp ([901](#)), StrComp ([897](#)), StrIComp ([900](#))

strlower

Synopsis: Convert null-terminated string to all-lowercase.

Declaration: `function strlower(p: pchar) : pchar`

Visibility: default

Description: Converts P to an all-lowercase string. Returns P.

Errors: None.

See also: StrUpper ([907](#))

Listing: ./stringex/ex14.pp

```

Program Example14;

Uses strings;

{ Program to demonstrate the StrLower and StrUpper functions. }

```

Const

```
P1 : PChar = 'THIS IS AN UPPERCASE PCHAR STRING';
P2 : PChar = 'this is a lowercase string';
```

begin

```
  WriteLn ( 'Uppercase : ', StrUpper(P2));
  StrLower (P1);
  WriteLn ( 'Lowercase : ', P1);
```

end.**strmove**

Synopsis: Move a null-terminated string to new location.

Declaration: `function strmove(dest: pchar; source: pchar; l: SizeInt) : pchar`

Visibility: default

Description: Copies MaxLen characters from Source to Dest. No terminating null-character is copied. Returns Dest

Errors: None.

See also: StrLCopy ([902](#)), StrCopy ([897](#))

Listing: ./stringex/ex10.pp

Program Example10;

Uses strings;

{ Program to demonstrate the StrMove function. }

Const P1 : PCHAR = 'This is a pchar string.';

Var P2 : Pchar;

begin

```
P2:= StrAlloc (StrLen(P1)+1);
StrMove (P2,P1, StrLen(P1)+1); { P2:=P1 }
WriteLn ( 'P2 = ', P2);
```

end.**strnew**

Synopsis: Allocate room for new null-terminated string.

Declaration: `function strnew(p: pchar) : pchar`

Visibility: default

Description: Copies P to the Heap, and returns a pointer to the copy.

Errors: Returns Nil if no memory was available for the copy.

See also: StrCopy ([897](#)), StrDispose ([898](#))

Listing: ./stringex/ex16.pp

Program Example16;

Uses strings;

{ Program to demonstrate the StrNew function. }

Const P1 : PChar = 'This is a PChar string';

var P2 : PChar;

begin

 P2:=**StrNew** (P1);

If P1=P2 **then**

writeln ('This can''t be happening...')

else

writeln ('P2 : ',P2);

end.

strupas

Synopsis: Convert a null-terminated string to a shortstring.

Declaration: functionstrupas(p: pchar) : shortstring

Visibility: default

Description: Converts a null terminated string in P to a Pascal string, and returns this string. The string is truncated at 255 characters.

Errors: None.

See also: StrPCopy ([905](#))

Listing: ./stringex/ex3.pp

Program Example3;

Uses strings;

{ Program to demonstrate the StrPas function. }

Const P : PChar = 'This is a PCHAR string';

var S : **string**;

begin

 S:=**StrPas** (P);

Writeln ('S : ',S);

end.

strcpy

Synopsis: Copy a pascal string to a null-terminated string

Declaration: functionstrcpy(d: pchar;const s: String) : pchar

Visibility: default

Description: Converts the Pascal string in *Source* to a Null-terminated string, and copies it to *Dest*. *Dest* needs enough room to contain the string *Source*, i.e. `Length (Source) + 1` bytes.

Errors: No length checking is performed.

See also: `StrPas` ([905](#))

Listing: `./stringex/ex2.pp`

Program `Example2;`

Uses `strings;`

{ Program to demonstrate the StrPCopy function. }

Const `S = 'This is a normal string.';`

Var `P : Pchar;`

begin

`p:= StrAlloc (length(S)+1);`

`if StrPCopy (P,S)<>P then`

`Writeln ('This is impossible !! ')`

`else`

`writeln (P);`

end.

strpos

Synopsis: Search for a null-terminated substring in a null-terminated string

Declaration: `function strpos(str1: pchar;str2: pchar) : pchar`

Visibility: default

Description: Returns a pointer to the first occurrence of *S2* in *S1*. If *S2* does not occur in *S1*, returns `Nil`.

Errors: None.

See also: `StrScan` ([907](#)), `StrRScan` ([907](#))

Listing: `./stringex/ex15.pp`

Program `Example15;`

Uses `strings;`

{ Program to demonstrate the StrPos function. }

Const `P : PChar = 'This is a PChar string.';`

`S : Pchar = 'is';`

begin

`Writeln ('Position of ''is'' in P : ',longint(StrPos(P,S))-Longint(P));`

end.

strrscan

Synopsis: Find last occurrence of a character in a null-terminated string.

Declaration: `function strrscan(p: pchar; c: Char) : pchar`

Visibility: default

Description: Returns a pointer to the last occurrence of the character C in the null-terminated string P. If C does not occur, returns Nil.

For an example, see StrScan (907).

Errors: None.

See also: StrScan (907), StrPos (906)

strscan

Synopsis: Find first occurrence of a character in a null-terminated string.

Declaration: `function strscan(p: pchar; c: Char) : pchar`

Visibility: default

Description: Returns a pointer to the first occurrence of the character C in the null-terminated string P. If C does not occur, returns Nil.

Errors: None.

See also: StrRScan (907), StrPos (906)

Listing: ./stringex/ex13.pp

Program Example13;

Uses strings;

{ Program to demonstrate the StrScan and StrRScan functions. }

Const P : PChar = 'This is a PCHAR string.';
 S : Char = 's' ;

begin

WriteLn ('P, starting from first 's' : ', StrScan(P,s));

WriteLn ('P, starting from last 's' : ', StrRScan(P,s));

end.

strupper

Synopsis: Convert null-terminated string to all-uppercase

Declaration: `function strupper(p: pchar) : pchar`

Visibility: default

Description: Converts P to an all-uppercase string. Returns P.

For an example, see StrLower (903)

Errors: None.

See also: StrLower (903)

Reference for unit 'strutils'

Table 27.1: Used units by unit 'strutl's'

27.2 Constants, types and variables

```

AnsiResemblesProc : TCompareTextProc = @SoundexProc

```

```
Brackets = [ ' ( , ' ) ' , ' [ , ' ] ' , ' { , ' } ' ]
```

Set of characters that contain all possible bracket characters

```
DigitChars = ['0'..'9']
```

Set of digit characters

```
StdSwitchChars = ['-','/']
```

Standard characters for the SwitchChars argument of GetCmdLineArg (920).

```
StdWordDelims = [#0..' ', '/', '.', ';', '/', '\', ':', ',', '"', '`'] + Brackets
```

Standard word delimiter values.

```
WordDelimiters : Set of Char = [#0..#255] - ['a'..'z', 'A'..'Z', '1'..'9', '0']
```

Standard word delimiters, used in the SearchBuf (927) call.

Types

`TCompareTextProc` = function(const AText: String;const AOther: String)
: Boolean

Function prototype for comparing two string in `AnsiResemblesText` (914)

`TSoundexIntLength` = 1..8

Range of allowed integer soundex lengths.

`TSoundexLength` = 1..MaxInt

Range of allowed soundex lengths.

`TStringSeachOption` = `TStringSearchOption`

There is an typo error in the original Borland `StrUtils` unit. This type just refers to the correct `TStringSearchOption` (909) and is provided for compatibility only.

`TStringSearchOption` = (soDown,soMatchCase,soWholeWord)

Table 27.2: Enumeration values for type `TStringSearchOption`

Value	Explanation
soDown	Search in down direction.
soMatchCase	Match case
soWholeWord	Search whole words only.

Possible options for `SearchBuf` (927) call.

`TStringSearchOptions`= Set of (soDown,soMatchCase,soWholeWord)

Set of options for `SearchBuf` (927) call.

27.3 Procedures and functions

AddChar

Synopsis: Add characters to the left of a string till a certain length

Declaration: function AddChar(C: Char;const S: String;N: Integer) : String

Visibility: default

Description: `AddChar` adds characters (C) to the left of S till the length N is reached, and returns the resulting string. If the length of S is already equal to or larger than N , then no characters are added. The resulting string can be thought of as a right-aligned version of S , with length N .

Errors: None

See also: `AddCharR` (910), `PadLeft` (925), `PadRight` (925), `PadCenter` (925)

AddCharR

Synopsis: Add chars at the end of a string till it reaches a certain length

Declaration: `function AddCharR(C: Char; const S: String; N: Integer) : String`

Visibility: default

Description: `AddCharR` adds characters (`C`) to the right of `S` till the length `N` is reached, and returns the resulting string. If the length of `S` is already equal to or larger than `N`, then no characters are added. The resulting string can be thought of as a left-aligned version of `S`, with length `N`.

Errors: None

See also: `AddChar` ([909](#))

AnsiContainsStr

Synopsis: Checks whether a string contains a given substring

Declaration: `function AnsiContainsStr(const AText: String; const ASubText: String) : Boolean`

Visibility: default

Description: `AnsiContainsString` checks whether `AText` contains `ASubText`, and returns `True` if this is the case, or returns `False` otherwise. The search is performed case-sensitive.

Errors: None

See also: `AnsiContainsText` ([910](#)), `AnsiEndsStr` ([910](#)), `AnsiIndexStr` ([911](#)), `AnsiStartsStr` ([914](#))

AnsiContainsText

Synopsis: Check whether a string contains a certain substring, ignoring case.

Declaration: `function AnsiContainsText(const AText: String; const ASubText: String) : Boolean`

Visibility: default

Description: `AnsiContainsString` checks whether `AText` contains `ASubText`, and returns `True` if this is the case, or returns `False` otherwise. The search is performed case-insensitive.

Errors:

See also: `AnsiContainsStr` ([910](#)), `AnsiEndsText` ([911](#)), `AnsiIndexText` ([911](#)), `AnsiStartsText` ([915](#))

AnsiEndsStr

Synopsis: Check whether a string ends with a certain substring

Declaration: `function AnsiEndsStr(const ASubText: String; const AText: String) : Boolean`

Visibility: default

Description: `AnsiEndsStr` checks `AText` to see whether it ends with `ASubText`, and returns `True` if it does, `False` if not. The check is performed case-sensitive. Basically, it checks whether the position of `ASubText` equals the length of `AText` minus the length of `ASubText` plus one.

Errors: None.

See also: [AnsiEndsText \(911\)](#), [AnsiStartsStr \(914\)](#), [AnsiIndexStr \(911\)](#), [AnsiContainsStr \(910\)](#)

AnsiEndsText

Synopsis: Check whether a string ends with a certain substring, ignoring case.

Declaration: `function AnsiEndsText(const ASubText: String;const AText: String)
: Boolean`

Visibility: default

Description: `AnsiEndsStr` checks `AText` to see whether it ends with `ASubText`, and returns `True` if it does, `False` if not. The check is performed case-insensitive. Basically, it checks whether the position of `ASubText` equals the length of `AText` minus the length of `ASubText` plus one.

Errors: None

See also: [AnsiStartsText \(915\)](#), [AnsiEndsStr \(910\)](#), [AnsiIndexText \(911\)](#), [AnsiContainsText \(910\)](#)

AnsiIndexStr

Synopsis: Searches, observing case, for a string in an array of strings.

Declaration: `function AnsiIndexStr(const AText: String;
const AValues: Array[] of String) : Integer`

Visibility: default

Description: `AnsiIndexStr` matches `AText` against each string in `AValues`. If a match is found, the corresponding index (zero-based) in the `AValues` array is returned. If no match is found, -1 is returned. The strings are matched observing case.

Errors: None.

See also: [AnsiIndexText \(911\)](#), [AnsiMatchStr \(912\)](#), [AnsiMatchText \(912\)](#)

AnsiIndexText

Synopsis: Searches, case insensitive, for a string in an array of strings.

Declaration: `function AnsiIndexText(const AText: String;
const AValues: Array[] of String) : Integer`

Visibility: default

Description: `AnsiIndexStr` matches `AText` against each string in `AValues`. If a match is found, the corresponding index (zero-based) in the `AValues` array is returned. If no match is found, -1 is returned. The strings are matched ignoring case.

Errors: None

See also: [AnsiIndexStr \(911\)](#), [AnsiMatchStr \(912\)](#), [AnsiMatchText \(912\)](#)

AnsiLeftStr

Synopsis: Copies a number of characters starting at the left of a string

Declaration: `function AnsiLeftStr(const AText: AnsiString; const ACount: Integer)
: AnsiString`

Visibility: default

Description: `AnsiLeftStr` returns the `ACount` leftmost characters from `AText`. If `ACount` is larger than the length of `AText`, only as much characters as available in `AText` will be copied. If `ACount` is zero or negative, no characters will be copied. The characters are counted as characters, not as Bytes.

This function corresponds to the Visual Basic `LeftStr` function.

Errors: None.

See also: `AnsiMidStr` (912), `AnsiRightStr` (914), `LeftStr` (923), `RightStr` (927), `MidStr` (924), `LeftBStr` (923), `RightBStr` (926), `MidBStr` (923)

AnsiMatchStr

Synopsis: Check whether a string occurs in an array of strings, observing case.

Declaration: `function AnsiMatchStr(const AText: String;
const AValues: Array[] of String) : Boolean`

Visibility: default

Description: `AnsiIndexStr` matches `AText` against each string in `AValues`. If a match is found, it returns `True`, otherwise `False` is returned. The strings are matched observing case.

This function simply calls `AnsiIndexStr` (911) and checks whether it returns -1 or not.

Errors:

AnsiMatchText

Synopsis: Check whether a string occurs in an array of strings, disregarding case.

Declaration: `function AnsiMatchText(const AText: String;
const AValues: Array[] of String) : Boolean`

Visibility: default

Description: `AnsiIndexStr` matches `AText` against each string in `AValues`. If a match is found, it returns `True`, otherwise `False` is returned. The strings are matched ignoring case.

This function simply calls `AnsiIndexText` (911) and checks whether it returns -1 or not.

Errors:

AnsiMidStr

Synopsis: Returns a number of characters copied from a given location in a string

Declaration: `function AnsiMidStr(const AText: AnsiString; const AStart: Integer;
const ACount: Integer) : AnsiString`

Visibility: default

Description: `AnsiMidStr` returns `ACount` characters from `AText`, starting at position `AStart`. If `AStart+ACount` is larger than the length of `AText`, only as much characters as available in `AText` (starting from `AStart`) will be copied. If `ACount` is zero or negative, no characters will be copied. The characters are counted as characters, not as Bytes.

This function corresponds to the Visual Basic `MidStr` function.

Errors: None

See also: `AnsiLeftStr` (912), `AnsiRightStr` (914), `LeftStr` (923), `RightStr` (927), `MidStr` (924), `LeftBStr` (923), `RightBStr` (926), `MidBStr` (923)

AnsiProperCase

Synopsis: Pretty-Print a string: make lowercase and capitalize first letters of words

Declaration: `function AnsiProperCase(const S: String;const WordDelims: TSysCharSet) : String`

Visibility: default

Description: `AnsiProperCase` converts `S` to an all lowercase string, but capitalizes the first letter of every word in the string, and returns the resulting string. When searching for words, the characters in `WordDelimiters` are used to determine the boundaries of words. The constant `StdWordDelims` (908) can be used for this.

Errors:

AnsiReplaceStr

Synopsis: Search and replace all occurrences of a string, case sensitive.

Declaration: `function AnsiReplaceStr(const AText: String;const AFromText: String;const AToText: String) : String`

Visibility: default

Description: `AnsiReplaceString` searches `AText` for all occurrences of the string `AFromText` and replaces them with `AToText`, and returns the resulting string. The search is performed observing case.

Errors: None.

See also: `AnsiReplaceText` (913), `SearchBuf` (927)

AnsiReplaceText

Synopsis: Search and replace all occurrences of a string, case sensitive.

Declaration: `function AnsiReplaceText(const AText: String;const AFromText: String;const AToText: String) : String`

Visibility: default

Description: `AnsiReplaceString` searches `AText` for all occurrences of the string `AFromText` and replaces them with `AToText`, and returns the resulting string. The search is performed ignoring case.

Errors: None.

See also: `AnsiReplaceStr` (913), `SearchBuf` (927)

AnsiResemblesText

Synopsis: Check whether 2 strings resemble each other.

Declaration: `function AnsiResemblesText(const AText: String;const AOther: String)
: Boolean`

Visibility: default

Description: `AnsiResemblesText` will check whether `AnsiResemblesProc` (908) is set. If it is not set, `False` is returned. If it is set, `AText` and `AOtherText` are passed to it and its result is returned.

Errors: None.

See also: `AnsiResemblesProc` (908), `SoundexProc` (929)

AnsiReverseString

Synopsis: Reverse the letters in a string.

Declaration: `function AnsiReverseString(const AText: AnsiString) : AnsiString`

Visibility: default

Description: `AnsiReverseString` returns a string with all characters of `AText` in reverse order.
if the result of this function equals `AText`, `AText` is called an anagram.

Errors: None.

AnsiRightStr

Synopsis: Copies a number of characters starting at the right of a string

Declaration: `function AnsiRightStr(const AText: AnsiString;const ACount: Integer)
: AnsiString`

Visibility: default

Description: `AnsiLeftStr` returns the `ACount` rightmost characters from `AText`. If `ACount` is larger than the length of `AText`, only as much characters as available in `AText` will be copied. If `ACount` is zero or negative, no characters will be copied. The characters are counted as characters, not as Bytes.
This function corresponds to the Visual Basic `RightStr` function.

Errors: None.

See also: `AnsiLeftStr` (912), `AnsiMidStr` (912), `LeftStr` (923), `RightStr` (927), `MidStr` (924), `LeftBStr` (923), `RightBStr` (926), `MidBStr` (923)

AnsiStartsStr

Synopsis: Check whether a string starts with a given substring, observing case

Declaration: `function AnsiStartsStr(const ASubText: String;const AText: String)
: Boolean`

Visibility: default

Description: `AnsiEndsStr` checks `AText` to see whether it starts with `ASubText` , and returns `True` if it does, `False` if not. The check is performed case-sensitive. Basically, it checks whether the position of `ASubText` equals 1.

Errors:

See also: `AnsiEndsStr` (910), `AnsiStartsStr` (914), `AnsiIndexStr` (911), `AnsiContainsStr` (910)

AnsiStartsText

Synopsis: Check whether a string starts with a given substring, ignoring case

Declaration: `function AnsiStartsText(const ASubText: String;const AText: String) : Boolean`

Visibility: default

Description: `AnsiEndsText` checks `AText` to see whether it starts with `ASubText` , and returns `True` if it does, `False` if not. The check is performed case-insensitive. Basically, it checks whether the position of `ASubText` equals 1.

Errors: None.

See also: `AnsiEndsText` (911), `AnsiStartsStr` (914), `AnsiIndexText` (911), `AnsiContainsText` (910)

Copy2Space

Synopsis: Returns all characters in a string till the first space character (not included).

Declaration: `function Copy2Space(const S: String) : String`

Visibility: default

Description: `Copy2Space` determines the position of the first space in the string `S` and returns all characters up to this position. The space character itself is not included in the result string. The string `S` is left untouched. If there is no space in `S`, then the whole string `S` is returned.

This function simply calls `Copy2Symb` (916) with the space (ASCII code 32) as the symbol argument.

Errors: None.

See also: `Copy2Symb` (916), `Copy2SpaceDel` (915)

Copy2SpaceDel

Synopsis: Deletes and returns all characters in a string till the first space character (not included).

Declaration: `function Copy2SpaceDel(var S: String) : String`

Visibility: default

Description: `Copy2SpaceDel` determines the position of the first space in the string `S` and returns all characters up to this position. The space character itself is not included in the result string. All returned characters, including the space, are deleted from the string `S`, after which it is right-trimmed. If there is no space in `S`, then the whole string `S` is returned, and `S` itself is emptied.

This function simply calls `Copy2SymbDel` (916) with the space (ASCII code 32) as the symbol argument.

Errors: None.

See also: [Copy2SymbDel \(916\)](#), [Copy2Space \(915\)](#)

Copy2Symb

Synopsis: Returns all characters in a string till a given character (not included).

Declaration: `function Copy2Symb(const S: String;Symb: Char) : String`

Visibility: default

Description: `Copy2SymbDel` determines the position of the first occurrence of `Symb` in the string `S` and returns all characters up to this position. The `Symb` character itself is not included in the result string. The string `S` is left untouched. If `Symb` does not appear in `S`, then the whole of `S` is returned.

Errors: None.

See also: [Copy2Space \(915\)](#), [Copy2SymbDel \(916\)](#)

Copy2SymbDel

Synopsis: Deletes and returns all characters in a string till a given character (not included).

Declaration: `function Copy2SymbDel(var S: String;Symb: Char) : String`

Visibility: default

Description: `Copy2SymbDel` determines the position of the first occurrence of `Symb` in the string `S` and returns all characters up to this position. The `Symb` character itself is not included in the result string. All returned characters, including the `Symb` character, are deleted from the string `S`, after which it is right-trimmed. If `Symb` does not appear in `S`, then the whole of `S` is returned, and `S` itself is emptied.

Errors: None.

See also: [Copy2SpaceDel \(915\)](#), [Copy2Symb \(916\)](#)

Dec2Numb

Synopsis: Convert a decimal number to a string representation, using given a base.

Declaration: `function Dec2Numb(N: LongInt;Len: Byte;Base: Byte) : String`

Visibility: default

Description: `Dec2Numb` converts `N` to its representation using base `Base`. The resulting string is left-padded with zeroes till it has length `Len`. `Base` must be in the range 2-36 to be meaningful, but no checking on this is performed.

Errors: If `Base` is out of range, the resulting string will contain unreadable (non-alphanumeric) characters.

See also: [Hex2Dec \(921\)](#), [IntToBin \(921\)](#), [intToRoman \(921\)](#), [RomanToInt \(927\)](#)

DecodeSoundexInt

Synopsis: Decodes the integer representation of a soundex code and returns the original soundex code.

Declaration: `function DecodeSoundexInt(AValue: Integer) : String`

Visibility: default

Description: `DecodeSoundexInt` converts the integer value `AValue` to a soundex string. It performs the reverse operation of the `SoundexInt` (929) function. The result is the soundex string corresponding to `AValue`.

Errors: None.

See also: `SoundexInt` (929), `DecodeSoundexWord` (917), `Soundex` (928)

DecodeSoundexWord

Synopsis: Decodes the word-sized representation of a soundex code and returns the original soundex code.

Declaration: `function DecodeSoundexWord(AValue: Word) : String`

Visibility: default

Description: `DecodeSoundexWord` converts the integer value `AValue` to a soundex string. It performs the reverse operation of the `SoundexWord` (930) function. The result is the soundex string corresponding to `AValue`.

Errors: None.

See also: `SoundexInt` (929), `DecodeSoundexInt` (917), `Soundex` (928)

DelChars

Synopsis: Delete all occurrences of a given character from a string.

Declaration: `function DelChars(const S: String; Chr: Char) : String`

Visibility: default

Description: `DelChars` returns a copy of `S` with all `Chr` characters removed from it.

Errors: None.

See also: `DelSpace` (917), `DelSpace1` (918)

DelSpace

Synopsis: Delete all occurrences of a space from a string.

Declaration: `function DelSpace(const S: String) : String`

Visibility: default

Description: `DelSpace` returns a copy of `S` with all spaces (ASCII code 32) removed from it.

Errors: None.

See also: `DelChars` (917), `DelSpace1` (918)

DelSpace1

Synopsis: Reduces sequences of space characters to 1 space character.

Declaration: `function DelSpace1(const S: String) : String`

Visibility: default

Description: `DelSpace1` returns a copy of `S` with all sequences of spaces reduced to 1 space.

Errors: None.

See also: `DelChars` (917), `DelSpace` (917)

DupeString

Synopsis: Creates and concatenates `N` copies of a string

Declaration: `function DupeString(const AText: String; ACount: Integer) : String`

Visibility: default

Description: `DupeString` returns a string consisting of `ACount` concatenations of `AText`. Thus

```
DupeString('1234567890',3);
```

will produce a string

```
'123456789012345678901234567890'
```

Errors: None.

ExtractDelimited

Synopsis: Extract the `N`-th delimited part from a string.

Declaration: `function ExtractDelimited(N: Integer; const S: String;
const Delims: TSysCharSet) : String`

Visibility: default

Description: `ExtractDelimited` extracts the `N`-th part from the string `S`. The set of characters in `Delims` are used to mark part boundaries. When a delimiter is encountered, a new part is started and the old part is ended. Another way of stating this is that any (possibly empty) series of characters not in `Delims`, situated between 2 characters in `Delims`, it is considered as piece of a part. This means that if 2 delimiter characters appear next to each other, there is an empty part between it. If an `N`-th part cannot be found, an empty string is returned. However, unlike `ExtractWord` (919), an empty string is a valid return value, i.e. a part can be empty.

The pre-defined constant `StdWordDelims` (908) can be used for the `Delims` argument. The pre-defined constant `Brackets` (908) would be better suited the `Delims` argument e.g. in case factors in a mathematical expression are searched.

Errors: None.

See also: `ExtractSubStr` (919), `ExtractWord` (919), `ExtractWordPos` (919)

ExtractSubstr

Synopsis: Extract a word from a string, starting at a given position in the string.

Declaration: `function ExtractSubstr(const S: String; var Pos: Integer;
const Delims: TSysCharSet) : String`

Visibility: default

Description: `ExtractSubStr` returns all characters from `S` starting at position `Pos` till the first character in `Delims`, or till the end of `S` is reached. The delimiter character is not included in the result. `Pos` is then updated to point to the next first non-delimiter character in `S`. If `Pos` is larger than the Length of `S`, an empty string is returned.

The pre-defined constant `StdWordDelims` (908) can be used for the `Delims` argument.

Errors: None.

See also: `ExtractDelimited` (918), `ExtractWord` (919), `ExtractWordPos` (919)

ExtractWord

Synopsis: Extract the N-th word out of a string.

Declaration: `function ExtractWord(N: Integer; const S: String;
const WordDelims: TSysCharSet) : String`

Visibility: default

Description: `ExtractWord` extracts the N-th word from the string `S`. The set of characters in `WordDelims` are used to mark word boundaries. A word is defined as any non-empty sequence of characters which are not present in `WordDelims`: if a character is not in `WordDelims`, it is considered as part of a word. If an N-th word cannot be found, an empty string is returned.

Unlike `ExtractDelimited` (918), an empty string is not a valid return value, i.e. is not a word. If an empty string is returned, the index `N` was out of range.

The pre-defined constant `StdWordDelims` (908) can be used for the `WordDelims` argument.

Errors: None.

See also: `ExtractWordPos` (919), `ExtractSubStr` (919), `ExtractDelimited` (918), `IsWordPresent` (922), `WordCount` (930), `WordPosition` (931)

ExtractWordPos

Synopsis: Extract a word from a string, and return the position where it was located in the string.

Declaration: `function ExtractWordPos(N: Integer; const S: String;
const WordDelims: TSysCharSet; var Pos: Integer)
: String`

Visibility: default

Description: `ExtractWordPos` extracts the N-th word from the string `S` and returns the position of this word in `Pos`. The set of characters in `WordDelims` are used to mark word boundaries. A word is defined as any non-empty sequence of characters which are not present in `WordDelims`: if a character is not in `WordDelims`, it is considered as part of a word. If an N-th word cannot be found, an empty string is returned and `Pos` is zero.

Unlike `ExtractDelimited` (918), an empty string is not a valid return value, i.e. is not a word. If an empty string is returned, the index `N` was out of range.

The pre-defined constant `StdWordDelims` (908) can be used for the `WordDelims` argument.

Errors: None.

See also: `ExtractWord` (919), `ExtractSubStr` (919), `IsWordPresent` (922), `WordCount` (930), `WordPosition` (931)

FindPart

Synopsis: Search for a substring in a string, using wildcards.

Declaration: `function FindPart(const HelpWilds: String; const InputStr: String)
: Integer`

Visibility: default

Description: `FindPart` searches the string `InputStr` and returns the first string that matches the wildcards specification in `HelpWilds`. If no match is found, an empty string is returned. valid wildcards are the "?" (question mark) and "*" (asterisk) characters.

Errors: None.

See also: `SearchBuf` (927)

GetCmdLineArg

Synopsis: Returns the command-line argument following the given switch.

Declaration: `function GetCmdLineArg(const Switch: String; SwitchChars: TSysCharSet)
: String`

Visibility: default

Description: `GetCmdLineArg` returns the value for the `Switch` option on the command-line, if any is given. Command-line arguments are considered switches if they start with one of the characters in the `SwitchChars` set. The value is the command-line argument following the switch command-line argument.

Gnu-style (long) Options of the form `switch=value` are not supported.

The `StdSwitchChars` (908) constant can be used as value for the `SwitchChars` parameter.

Errors: The `GetCmdLineArg` does not check whether the value of the option does not start with a switch character. i.e.

```
myprogram -option1 -option2
```

will result in "-option2" as the result of the `GetCmdLineArg` call for `option1`.

See also: `StdSwitchChars` (908)

Hex2Dec

Synopsis: Converts a hexadecimal string to a decimal value

```
Declaration: function Hex2Dec(const S: String) : LongInt
```

Visibility: default

Description: Hex2Dec converts the hexadecimal value in the string S to its decimal value. Unlike the standard Val or StrToInt functions, there need not be a \$ sign in front of the hexadecimal value to indicate that it is indeed a hexadecimal value.

Errors: If `S` does not contain a valid hexadecimal value, an `EConvertError` exception will be raised.

See also: [Dec2Numb \(916\)](#), [IntToBin \(921\)](#), [intToRoman \(921\)](#), [RomanToInt \(927\)](#)

IfThen

Synopsis: Returns one of two strings, depending on a boolean expression

```
Declaration: function IfThen(AValue: Boolean; const ATrue: String; AFalse: String)
              : String
              function IfThen(AValue: Boolean; const ATrue: String) : String
```

Visibility: default

Description: IfThen returns ATrue if AValue is True, and returns AFalse if AValue is false.

Errors: None.

See also: [AnsiMatchStr \(912\)](#), [AnsiMatchText \(912\)](#)

IntToBin

Synopsis: Converts an integer to a binary string representation, inserting spaces at fixed locations.

```
Declaration: function IntToBin(Value: LongInt; Digits: Integer; Spaces: Integer)
                : String
```

Visibility: default

Description: `IntToBin` converts `Value` to a string with it's binary (base 2) representation. The resulting string contains at least `Digits` digits, with spaces inserted every `Spaces` digits. `Spaces` should be a nonzero value. If `Digits` is larger than 32, it is truncated to 32.

Errors: If spaces is zero, a division by zero error will occur.

See also: [Hex2Dec \(921\)](#), [IntToRoman \(921\)](#)

IntToRoman

Synopsis: Represent an integer with roman numerals

Declaration: `function IntToRoman(Value: LongInt) : String`

Visibility: default

Description: `IntToRoman` converts `Value` to a string with the Roman representation of `Value`. Number up to 1 million can be represented this way.

Errors: None.

See also: [RomanToInt \(927\)](#), [Hex2Dec \(921\)](#), [IntToBin \(921\)](#)

IsEmptyStr

Synopsis: Check whether a string is empty, disregarding whitespace characters

Declaration: `function IsEmptyStr(const S: String; const EmptyChars: TSysCharSet)
: Boolean`

Visibility: default

Description: `IsEmptyStr` returns `True` if the string `S` only contains characters whitespace characters, all characters in `EmptyChars` are considered whitespace characters. If a character not present in `EmptyChars` is found in `S`, `False` is returned.

Errors: None.

See also: `IsWild` ([922](#)), `FindPart` ([920](#)), `IsWordPresent` ([922](#))

IsWild

Synopsis: Check whether a string matches a wildcard search expression.

Declaration: `function IsWild(InputStr: String; Wilds: String; IgnoreCase: Boolean)
: Boolean`

Visibility: default

Description: `IsWild` checks `InputStr` for the presence of the `Wilds` string. `Wilds` may contain "?" and "*" wildcard characters, which have their usual meaning: "*" matches any series of characters, possibly empty. "?" matches any single character. The function returns `True` if a string is found that matches `Wilds`, `False` otherwise.

If `IgnoreCase` is `True`, the non-wildcard characters are matched case insensitively. If it is `False`, case is observed when searching.

Errors: None.

See also: `SearchBuf` ([927](#)), `FindPart` ([920](#))

IsWordPresent

Synopsis: Check for the presence of a word in a string.

Declaration: `function IsWordPresent(const W: String; const S: String;
const WordDelims: TSysCharSet) : Boolean`

Visibility: default

Description: `IsWordPresent` checks for the presence of the word `W` in the string `S`. Words are delimited by the characters found in `WordDelims`. The function returns `True` if a match is found, `False` otherwise. The search is performed case sensitive.

This function is equivalent to the `SearchBuf` ([927](#)) function with the `soWholeWords` option specified.

Errors: None.

See also: `SearchBuf` ([927](#))

LeftBStr

Synopsis: Copies Count characters starting at the left of a string.

Declaration: `function LeftBStr(const AText: AnsiString; const AByteCount: Integer)
: AnsiString`

Visibility: default

Description: `LeftBStr` returns a string containing the leftmost `AByteCount` bytes from the string `AText` .
If `AByteCount` is larger than the length (in bytes) of `AText` , only as many bytes as available are returned.

Errors: None.

See also: `LeftStr` ([923](#)), `AnsiLeftStr` ([912](#)), `RightBStr` ([926](#)), `MidBStr` ([923](#))

LeftStr

Synopsis: Copies Count characters starting at the left of a string.

Declaration: `function LeftStr(const AText: AnsiString; const ACount: Integer)
: AnsiString`
`function LeftStr(const AText: WideString; const ACount: Integer)
: WideString`

Visibility: default

Description: `LeftStr` returns a string containing the leftmost `ACount` characters from the string `AText` . If
`ACount` is larger than the length (in characters) of `AText` , only as many characters as available are returned.

Errors: None.

See also: `LeftBStr` ([923](#)), `AnsiLeftStr` ([912](#)), `RightStr` ([927](#)), `MidStr` ([924](#))

MidBStr

Synopsis: Copies a number of characters starting at a given position in a string.

Declaration: `function MidBStr(const AText: AnsiString; const AByteStart: Integer;
const AByteCount: Integer) : AnsiString`

Visibility: default

Description: `MidBStr` returns a string containing the first `AByteCount` bytes from the string `AText` starting
at position `AByteStart`. If `AByteStart+AByteCount` is larger than the length (in bytes) of
`AText`, only as many bytes as available are returned. If `AByteStart` is less than 1 or larger than
the length of `AText`, then no characters are returned.

Errors: None.

See also: `LeftBStr` ([923](#)), `AnsiMidStr` ([912](#)), `RightBStr` ([926](#)), `MidStr` ([924](#))

MidStr

Synopsis: Copies a number of characters starting at a given position in a string.

Declaration: `function MidStr(const AText: AnsiString;const AStart: Integer;
const ACount: Integer) : AnsiString
function MidStr(const AText: WideString;const AStart: Integer;
const ACount: Integer) : WideString`

Visibility: default

Description: `MidStr` returns a string containing the first `ACount` bytes from the string `AText` starting at position `AStart`. If `AStart+ACount` is larger than the length (in characters) of `AText`, only as many characters as available are returned. If `AStart` is less than 1 or larger than the length of `AText`, then no characters are returned.

This function is equivalent to the standard `Copy` function, and is provided for completeness only.

Errors: None.

See also: [LeftStr \(923\)](#), [AnsiMidStr \(912\)](#), [RightStr \(927\)](#), [MidBStr \(923\)](#)

NPos

Synopsis: Returns the position of the N-th occurrence of a substring in a string.

Declaration: `function NPos(const C: String;S: String;N: Integer) : Integer`

Visibility: default

Description: `NPos` checks `S` for the position of the N-th occurrence of `C`. If `C` occurs less than `N` times in `S`, or does not occur in `S` at all, 0 is returned. If `N` is less than 1, zero is returned.

Errors: None.

See also: [WordPosition \(931\)](#), [FindPart \(920\)](#)

Numb2Dec

Synopsis: Converts a string representation of a number to its numerical value, given a certain base.

Declaration: `function Numb2Dec(S: String;Base: Byte) : LongInt`

Visibility: default

Description: `Numb2Dec` converts the number in string `S` to a decimal value. It assumes the number is represented using `Base` as the base. No checking is performed to see whether `S` contains a valid number using base `Base`.

Errors: None.

See also: [Hex2Dec \(921\)](#), [Numb2USA \(925\)](#)

Numb2USA

Synopsis: Insert thousand separators.

Declaration: `function Numb2USA(const S: String) : String`

Visibility: default

Description: `Numb2USA` inserts thousand separators in the string `S` at the places where they are supposed to be, i.e. every 3 digits. The string `S` should contain a valid integer number, i.e. no digital number. No checking on this is done.

Errors: None.

PadCenter

Synopsis: Pad the string to a certain length, so the string is centered.

Declaration: `function PadCenter(const S: String;Len: Integer) : String`

Visibility: default

Description: `PadCenter` add spaces to the left and right of the string `S` till the result reaches length `Len`. If the number of spaces to add is odd, then the extra space will be added at the end. If the string `S` has length equal to or largert than `Len`, no spaces are added, and the string `S` is returned as-is.

Errors: None.

See also: `PadLeft` ([925](#)), `PadRight` ([925](#)), `AddChar` ([909](#)), `AddCharR` ([910](#))

PadLeft

Synopsis: Add spaces to the left of a string till a certain length is reached.

Declaration: `function PadLeft(const S: String;N: Integer) : String`

Visibility: default

Description: `PadLeft` add spaces to the left of the string `S` till the result reaches length `Len` . If the string `S` has length equal to or largert than `Len` , no spaces are added, and the string `S` is returned as-is. The resulting string is `S` , right-justified on length `Len` .

Errors: None.

See also: `PadLeft` ([925](#)), `PadCenter` ([925](#)), `AddChar` ([909](#)), `AddCharR` ([910](#))

PadRight

Synopsis: Add spaces to the right of a string till a certain length is reached.

Declaration: `function PadRight(const S: String;N: Integer) : String`

Visibility: default

Description: `PadRight` add spaces to the left of the string `S` till the result reaches length `Len`. If the string `S` has length equal to or largert than `Len`, no spaces are added, and the string `S` is returned as-is. The resulting string is `S`, left-justified on length `Len` .

Errors: None.

See also: `PadLeft` ([925](#)), `PadCenter` ([925](#)), `AddChar` ([909](#)), `AddCharR` ([910](#))

PosEx

Synopsis: Search for the occurrence of a character in a string, starting at a certain position.

```
Declaration: function PosEx(const SubStr: String;const S: String;Offset: Cardinal)
              : Integer
              function PosEx(const SubStr: String;const S: String) : Integer
              function PosEx(c: Char;const S: String;Offset: Cardinal) : Integer
```

Visibility: default

Description: `PosEx` returns the position of the first occurrence of the character `C` or the substring `SubStr` in the string `S`, starting the search at position `Offset` (default 1). If `C` or `SubStr` does not occur in `S` after the given `Offset`, zero is returned. The position `Offset` is also searched.

Errors: None.

See also: [NPos \(924\)](#), [AnsiContainsText \(910\)](#), [AnsiContainsStr \(910\)](#)

RandomFrom

Synopsis: Choose a random string from an array of strings.

```
Declaration: function RandomFrom(const AValues: Array[] of String) : String
                                ; Overload
```

Visibility: default

Description: RandomFrom picks at random a valid index in the array AValues and returns the string at that position in the array.

Errors: None.

See also: [AnsiMatchStr \(912\)](#), [AnsiMatchText \(912\)](#)

ReverseString

Synopsis: Reverse characters in a string

Declaration: `function ReverseString(const AText: String) : String`

Visibility: default

Description: ReverseString returns a string, made up of the characters in string AText, in reverse order.

Errors: None.

See also: [RandomFrom \(926\)](#)

RightBStr

Synopsis: Copy a given number of characters (bytes), counting from the right of a string.

```
Declaration: function RightBStr(const AText: AnsiString; const AByteCount: Integer)
                : AnsiString
```

Visibility: default

Description: `RightBStr` returns a string containing the rightmost `AByteCount` bytes from the string `AText`. If `AByteCount` is larger than the length (in bytes) of `AText`, only as many bytes as available are returned.

Errors: None.

See also: `LeftBStr` ([923](#)), `AnsiRightStr` ([914](#)), `RightStr` ([927](#)), `MidBStr` ([923](#))

RightStr

Synopsis: Copy a given number of characters, counting from the right of a string.

Declaration:

```
function RightStr(const AText: AnsiString; const ACount: Integer)
    : AnsiString
function RightStr(const AText: WideString; const ACount: Integer)
    : WideString
```

Visibility: default

Description: `RightStr` returns a string containing the rightmost `ACount` characters from the string `AText`. If `ACount` is larger than the length (in characters) of `AText`, only as many characters as available are returned.

Errors: None.

See also: `LeftStr` ([923](#)), `AnsiRightStr` ([914](#)), `RightBStr` ([926](#)), `MidStr` ([924](#))

RomanToInt

Synopsis: Convert a string with a Roman number to its decimal value.

Declaration:

```
function RomanToInt(const S: String) : LongInt
```

Visibility: default

Description: `RomanToInt` returns the decimal equivalent of the Roman numerals in the string `S`. Invalid characters are dropped from `S`. A negative numeral is supported as well.

Errors: None.

See also: `IntToRoman` ([921](#)), `Hex2Dec` ([921](#)), `Numb2Dec` ([924](#))

SearchBuf

Synopsis: Search a buffer for a certain string.

Declaration:

```
function SearchBuf(Buf: PChar; BufLen: Integer; SelStart: Integer;
    SelLength: Integer; SearchString: String;
    Options: TStringSearchOptions) : PChar
function SearchBuf(Buf: PChar; BufLen: Integer; SelStart: Integer;
    SelLength: Integer; SearchString: String) : PChar
```

Visibility: default

Table 27.3:

Option	Effect
soDown	Searches forward, starting at the end of the selection. Default is searching up
soMatchCase	Observe case when searching. Default is to ignore case.
soWholeWord	Match only whole words. Default also returns parts of words

Description: `SearchBuf` searches the buffer `Buf` for the occurrence of `SearchString`. At most `Buflen` characters are searched, and the search is started at `SelStart+SelLength`. If a match is found, a pointer to the position of the match is returned. The parameter `Options` (909) specifies how the search is conducted. It is a set of the following options:

The standard constant `WordDelimiters` (908) is used to mark the boundaries of words.

The `SelStart` parameter is zero based.

Errors: `Buflen` must be the real length of the string, no checking on this is performed.

See also: `FindPart` (920), `ExtractWord` (919), `ExtractWordPos` (919), `ExtractSubStr` (919), `IsWordPresent` (922)

Soundex

Synopsis: Compute the soundex of a string

Declaration: `function Soundex(const AText: String; ALength: TSoundexLength) : String`
`function Soundex(const AText: String) : String`

Visibility: default

Description: `Soundex` computes a soundex code for `AText`. The resulting code will at most have `ALength` characters. The soundex code is computed according to the US system of soundex computing, which may result in inaccurate results in other languages.

Errors: None.

See also: `SoundexCompare` (928), `SoundexInt` (929), `SoundexProc` (929), `SoundexWord` (930), `SoundexSimilar` (929)

SoundexCompare

Synopsis: Compare soundex values of 2 strings.

Declaration: `function SoundexCompare(const AText: String; const AOther: String; ALength: TSoundexLength) : Integer`
`function SoundexCompare(const AText: String; const AOther: String) : Integer`

Visibility: default

Description: `SoundexCompare` computes the soundex codes of `AText` and `AOther` and feeds these to `CompareText`. It will return -1 if the soundex code of `AText` is less than the soundex code of `AOther`, 0 if they are equal, and 1 if the code of `AOther` is larger than the code of `AText`.

Errors: None.

See also: `Soundex` (928), `SoundexInt` (929), `SoundexProc` (929), `SoundexWord` (930), `SoundexSimilar` (929)

SoundexInt

Synopsis: Soundex value as an integer.

Declaration: `function SoundexInt(const AText: String; ALength: TSoundexIntLength)
: Integer
function SoundexInt(const AText: String) : Integer`

Visibility: default

Description: `SoundexInt` computes the Soundex (928) code (with length `ALength`, default 4) of `AText`, and converts the code to an integer value.

Errors: None.

See also: Soundex (928), SoundexCompare (928), SoundexProc (929), SoundexWord (930), SoundexSimilar (929)

SoundexProc

Synopsis: Default `AnsiResemblesText` implementation.

Declaration: `function SoundexProc(const AText: String; const AOther: String) : Boolean`

Visibility: default

Description: `SoundexProc` is the standard implementation for the `AnsiResemblesText` (914) procedure: By default, `AnsiResemblesProc` is set to this function. It compares the soundex codes of `AOther` and `AText` and returns `True` if they are equal, or `False` if they are not.

Errors: None.

See also: Soundex (928), SoundexCompare (928), SoundexInt (929), SoundexWord (930), SoundexSimilar (929)

SoundexSimilar

Synopsis: Check whether 2 strings have equal soundex values

Declaration: `function SoundexSimilar(const AText: String; const AOther: String;
ALength: TSoundexLength) : Boolean
function SoundexSimilar(const AText: String; const AOther: String)
: Boolean`

Visibility: default

Description: `SoundexSimilar` returns `True` if the soundex codes (with length `ALength`) of `AText` and `AOther` are equal, and `False` if they are not.

Errors: None.

See also: Soundex (928), SoundexCompare (928), SoundexInt (929), SoundexProc (929), SoundexWord (930), Soundex (928)

SoundexWord

Synopsis: Calculate a word-sized soundex value

Declaration: `function SoundexWord(const AText: String) : Word`

Visibility: default

Description: `SoundexInt` computes the Soundex (928) code (with length 4) of `AText` , and converts the code to a word-sized value.

Errors: None.

See also: Soundex (928), SoundexCompare (928), SoundexInt (929), SoundexProc (929), SoundexSimilar (929)

StuffString

Synopsis: Replace part of a string with another string.

Declaration: `function StuffString(const AText: String; AStart: Cardinal;
 ALength: Cardinal; const ASubText: String) : String`

Visibility: default

Description: `StuffString` returns a copy of `AText` with the segment starting at `AStart` with length `ALength`, replaced with the string `ASubText`. Basically it deletes the segment of `AText` and inserts the new text in it's place.

Errors: No checking on the validity of the `AStart` and `ALength` parameters is done. Providing invalid values may result in access violation errors.

See also: FindPart (920), DelChars (917), DelSpace (917), ExtractSubStr (919), DupeString (918)

Tab2Space

Synopsis: Convert tab characters to a number of spaces

Declaration: `function Tab2Space(const S: String; Numb: Byte) : String`

Visibility: default

Description: `Tab2Space` returns a copy of `S` with all tab characters (ASCII character 9) converted to `Numb` spaces.

Errors: None.

See also: StuffString (930), FindPart (920), ExtractWord (919), DelChars (917), DelSpace (917), DelSpace1 (918), DupeString (918)

WordCount

Synopsis: Count the number of words in a string.

Declaration: `function WordCount(const S: String; const WordDelims: TSysCharSet)
 : Integer`

Visibility: default

Description: `WordCount` returns the number of words in the string `S`. A word is a non-empty string of characters bounded by one of the characters in `WordDelims`.

The pre-defined `StdWordDelims` (908) constant can be used for the `WordDelims` argument.

Errors: None.

See also: `WordPosition` (931), `StdWordDelims` (908), `ExtractWord` (919), `ExtractWordPos` (919)

WordPosition

Synopsis: Search position of Nth word in a string.

Declaration: `function WordPosition(const N: Integer; const S: String;
const WordDelims: TSysCharSet) : Integer`

Visibility: default

Description: `WordPosition` returns the position (in characters) of the N-th word in the string `S`. A word is a non-empty string of characters bounded by one of the characters in `WordDelims`. If `N` is out of range, zero is returned.

The pre-defined `StdWordDelims` (908) constant can be used for the `WordDelims` argument.

Errors: None

See also: `WordCount` (930), `StdWordDelims` (908), `ExtractWord` (919), `ExtractWordPos` (919)

XorDecode

Synopsis: Decode a string encoded with `XorEncode` (931)

Declaration: `function XorDecode(const Key: String; const Source: String) : String`

Visibility: default

Description: `XorDecode` decodes `Source` and returns the original string that was encrypted using `XorEncode` (931) with key `Key`. If a different key is used than the key used to encode the string, the result will be unreadable.

Errors: If the string `Source` is not a valid `XorEncode` result (e.g. contains non-numerical characters), then a `EConversionError` exception will be raised.

See also: `XorEncode` (931), `XorString` (932)

XorEncode

Synopsis: Encode a string by XOR-ing its characters using characters of a given key, representing the result as hex values.

Declaration: `function XorEncode(const Key: String; const Source: String) : String`

Visibility: default

Description: `XorEncode` encodes the string `Source` by XOR-ing each character in `Source` with the corresponding character in `Key` (repeating `Key` as often as necessary) and representing the resulting ASCII code as a hexadecimal number (of length 2). The result is therefore twice as long as the original string, and every 2 bytes represent an ASCII code.

Feeding the resulting string with the same key `Key` to the `XorDecode` (931) function will result in the original `Source` string.

This function can be used e.g. to trivially encode a password in a configuration file.

Errors: None.

See also: `XorDecode` (931), `XorString` (932), `Hex2Dec` (921)

XorString

Synopsis: Encode a string by XOR-ing its characters using characters of a given key.

Declaration: `function XorString(const Key: ShortString; const Src: ShortString)
: ShortString`

Visibility: default

Description: `XorString` encodes the string `Src` by XOR-ing each character in `Source` with the corresponding character in `Key`, repeating `Key` as often as necessary. The resulting string may contain unreadable characters and may even contain null characters. For this reason it may be a better idea to use the `XorEncode` (931) function instead, which will represent each resulting ASCII code as a hexadecimal number (of length 2).

Feeding the result again to `XorString` with the same `Key`, will result in the original string `Src`.

Errors: None.

See also: `XorEncode` (931), `XorDecode` (931)

Chapter 28

Reference for unit 'System'

28.1 Miscellaneous functions

Functions that do not belong in one of the other categories.

Table 28.1:

Name	Description
Assert (977)	Conditionally abort program with error
Break (980)	Abort current loop
Continue (988)	Next cycle in current loop
Exclude (995)	Exclude an element from a set
Exit (997)	Exit current function or procedure
Include (1011)	Include an element into a set
LongJump (1019)	Jump to execution point
Ord (1024)	Return ordinal value of enumerated type
Pred (1027)	Return previous value of ordinal type
SetJump (1039)	Mark execution point for jump
SizeOf (1043)	Return size of variable or type
Succ (1048)	Return next value of ordinal type

28.2 Operating System functions

Functions that are connected to the operating system.

28.3 String handling

All things connected to string handling.

28.4 Mathematical routines

Functions connected to calculating and converting numbers.

Table 28.2:

Name	Description
Chdir (981)	Change working directory
Getdir (1004)	Return current working directory
Halt (1007)	Halt program execution
Paramcount (1025)	Number of parameters with which program was called
Paramstr (1025)	Retrieve parameters with which program was called
Mkdir (1021)	Make a directory
Rmdir (1035)	Remove a directory
Runerror (1036)	Abort program execution with error condition

Table 28.3:

Name	Description
BinStr (978)	Construct binary representation of integer
Chr (982)	Convert ASCII code to character
Concat (987)	Concatenate two strings
Copy (989)	Copy part of a string
Delete (991)	Delete part of a string
HexStr (1008)	Construct hexadecimal representation of integer
Insert (1015)	Insert one string in another
Length (1017)	Return length of string
Lowercase (1020)	Convert string to all-lowercase
OctStr (1023)	Construct octal representation of integer
Pos (1026)	Calculate position of one string in another
SetLength (1040)	Set length of a string
SetString (1041)	Set contents and length of a string
Str (1046)	Convert number to string representation
StringOfChar (1046)	Create string consisting of a number of characters
Uppcase (1053)	Convert string to all-uppercase
Val (1054)	Convert string to number

28.5 Memory management functions

Functions concerning memory issues.

28.6 File handling functions

Functions concerning input and output from and to file.

28.7 Variables

The following variables are defined and initialized in the system unit:

```
var
  output, input, stderr : text;
  exitproc : pointer;
```

Table 28.4:

Name	Description
Abs (972)	Calculate absolute value
Arctan (975)	Calculate inverse tangent
Cos (989)	Calculate cosine of angle
Dec (990)	Decrease value of variable
Exp (998)	Exponentiate
Frac (1003)	Return fractional part of floating point value
Hi (1008)	Return high byte/word of value
Inc (1010)	Increase value of variable
Int (1015)	Calculate integer part of floating point value
Ln (1018)	Calculate logarithm
Lo (1019)	Return low byte/word of value
Odd (1023)	Is a value odd or even ?
Pi (1026)	Return the value of pi
Power (1027)	Raise float to integer power
Random (1029)	Generate random number
Randomize (1030)	Initialize random number generator
Round (1035)	Round floating point value to nearest integer number
Sin (1043)	Calculate sine of angle
Sqr (1044)	Calculate the square of a value
Sqrt (1045)	Calculate the square root of a value
Swap (1048)	Swap high and low bytes/words of a variable
Trunc (1052)	Truncate a floating point value

```
exitcode : word;
stackbottom : Cardinal;
```

The variables `ExitProc`, `exitcode` are used in the Free Pascal exit scheme. It works similarly to the one in Turbo Pascal:

When a program halts (be it through the call of the `Halt` function or `Exit` or through a run-time error), the exit mechanism checks the value of `ExitProc`. If this one is non-`Nil`, it is set to `Nil`, and the procedure is called. If the exit procedure exits, the value of `ExitProc` is checked again. If it is non-`Nil` then the above steps are repeated. So when an exit procedure must be installed, the old value of `ExitProc` should be saved (it may be non-`Nil`, since other units could have set it). In the exit procedure the value of `ExitProc` should be restored to the previous value, such that if it was non-`Nil` the exit-procedure can be called.

The `ErrorAddr` and `ExitCode` can be used to check for error-conditions. If `ErrorAddr` is non-`Nil`, a run-time error has occurred. If so, `ExitCode` contains the error code. If `ErrorAddr` is `Nil`, then `{ExitCode}` contains the argument to `Halt` or 0 if the program terminated normally.

`ExitCode` is always passed to the operating system as the exit-code of the current process.

Remark: The maximum error code under linux and unix like operating systems is 127.

Under GO32, the following constants are also defined :

```
const
  seg0040 = $0040;
  segA000 = $A000;
  segB000 = $B000;
  segB800 = $B800;
```

Table 28.5:

Name	Description
Addr (974)	Return address of variable
Assigned (978)	Check if a pointer is valid
CompareByte (983)	Compare 2 memory buffers byte per byte
CompareChar (984)	Compare 2 memory buffers byte per byte
CompareDWord (985)	Compare 2 memory buffers byte per byte
CompareWord (986)	Compare 2 memory buffers byte per byte
CSeg (990)	Return code segment
Dispose (992)	Free dynamically allocated memory
DSeg (993)	Return data segment
FillByte (1000)	Fill memory region with 8-bit pattern
Fillchar (1000)	Fill memory region with certain character
FillDWord (1001)	Fill memory region with 32-bit pattern
Fillword (1002)	Fill memory region with 16-bit pattern
Freemem (1003)	Release allocated memory
Getmem (1005)	Allocate new memory
GetMemoryManager (1006)	Return current memory manager
High (1009)	Return highest index of open array or enumerated
IsMemoryManagerSet (1017)	Is the memory manager set
Low (1020)	Return lowest index of open array or enumerated
Move (1021)	Move data from one location in memory to another
MoveChar0 (1022)	Move data till first zero character
New (1022)	Dynamically allocate memory for variable
Ofs (1024)	Return offset of variable
Ptr (1028)	Combine segment and offset to pointer
ReAllocMem (1032)	Resize a memory block on the heap
Seg (1038)	Return segment
SetMemoryManager (1040)	Set a memory manager
Sptr (1044)	Return current stack pointer
SSeg (1045)	Return stack segment register value

These constants allow easy access to the bios/screen segment via mem/absolute.

The randomize function uses a seed stored in the RandSeed variable:

```
RandSeed      : Cardinal;
```

This variable is initialized in the initialization code of the system unit.

Other variables indicate the state of the application.

```
IsLibrary : boolean;
IsMultiThread : boolean;
```

The IsLibrary variable is set to true if this module is a shared library instead of an application. The IsMultiThread variable is set to True if the application has spawned other threads, otherwise, and by default, it is set to False.

28.8 Constants

The following constants define the maximum values that can be used with various types:

Table 28.6:

Name	Description
Append (975)	Open a file in append mode
Assign (977)	Assign a name to a file
Blockread (979)	Read data from a file into memory
Blockwrite (980)	Write data from memory to a file
Close (982)	Close a file
Eof (993)	Check for end of file
Eoln (994)	Check for end of line
Erase (995)	Delete file from disk
Filepos (998)	Position in file
Filesize (999)	Size of file
Flush (1002)	Write file buffers to disk
IOresult (1016)	Return result of last file IO operation
Read (1030)	Read from file into variable
Readln (1031)	Read from file into variable and goto next line
Rename (1033)	Rename file on disk
Reset (1033)	Open file for reading
Rewrite (1034)	Open file for writing
Seek (1036)	Set file position
SeekEof (1037)	Set file position to end of file
SeekEoln (1038)	Set file position to end of line
SetTextBuf (1041)	Set size of file buffer
Truncate (1052)	Truncate the file at position
Write (1054)	Write variable to file
WriteLn (1055)	Write variable to file and append newline

```

MaxSIntValue = High(ValSInt);
MaxUIntValue = High(ValUInt);
maxint      = maxsmallint;
maxLongint  = $7fffffff;
maxSmallint = 32767;

```

The following constants for file-handling are defined in the system unit:

```

Const
  fmclosed = $D7B0;
  fminput  = $D7B1;
  fmoutput = $D7B2;
  fminout  = $D7B3;
  fmappend = $D7B4;
  filemode : byte = 2;

```

The `filemode` variable is used when a non-text file is opened using `Reset`. It indicates how the file will be opened. `filemode` can have one of the following values:

- 0 The file is opened for reading.
- 1 The file is opened for writing.
- 2 The file is opened for reading and writing.

The default value is 2. Other values are possible but are operating system specific.

Further, the following non processor specific general-purpose constants are also defined:

```
const
  erroraddr : pointer = nil;
  errorcode : word = 0;
  { max level in dumping on error }
  max_frame_dump : word = 20;
```

Remark: Processor specific global constants are named Testxxxx where xxxx represents the processor number (such as Test8086, Test68000), and are used to determine on what generation of processor the program is running on.

The following constants are defined to access VMT entries:

```
vmtInstanceSize      = 0;
  vmtParent           = 8;
  vmtClassName        = 12;
  vmtDynamicTable     = 16;
  vmtMethodTable      = 20;
  vmtFieldTable       = 24;
  vmtTypeInfo         = 28;
  vmtInitTable        = 32;
  vmtAutoTable        = 36;
  vmtIntfTable        = 40;
  vmtMsgStrPtr        = 44;
  vmtMethodStart      = 48;
  vmtDestroy          = vmtMethodStart;
  vmtNewInstance      = vmtMethodStart+4;
  vmtFreeInstance     = vmtMethodStart+8;
  vmtSafeCallException = vmtMethodStart+12;
  vmtDefaultHandler   = vmtMethodStart+16;
  vmtAfterConstruction = vmtMethodStart+20;
  vmtBeforeDestruction = vmtMethodStart+24;
  vmtDefaultHandlerStr = vmtMethodStart+28;
```

The constant names should be used, and never their values, because the VMT table can change, breaking code that uses direct values.

The following constants will be used for the planned variant support:

```
varEmpty      = $0000;
  varNull      = $0001;
  varSmallint  = $0002;
  varInteger   = $0003;
  varSingle    = $0004;
  varDouble    = $0005;
  varCurrency  = $0006;
  varDate      = $0007;
  varOleStr    = $0008;
  varDispatch  = $0009;
  varError     = $000A;
  varBoolean   = $000B;
  varVariant   = $000C;
```

```

varUnknown    = $000D;
varByte       = $0011;
varString     = $0100;
varAny        = $0101;
varTypeMask   = $0FFF;
varArray      = $2000;
varByRef      = $4000;

```

The following constants are used in the TVarRec record:

```

vtInteger     = 0;
vtBoolean     = 1;
vtChar        = 2;
vtExtended    = 3;
vtString      = 4;
vtPointer     = 5;
vtPChar       = 6;
vtObject      = 7;
vtClass       = 8;
vtWideChar    = 9;
vtPWideChar   = 10;
vtAnsiString  = 11;
vtCurrency    = 12;
vtVariant     = 13;
vtInterface   = 14;
vtWideString  = 15;
vtInt64       = 16;
vtQWord       = 17;

```

The ExceptProc is called when an unhandled exception occurs:

```

Const
  ExceptProc : TExceptProc = Nil;

```

It is set in the objpas unit, but it can be set by the programmer to change the default exception handling.

The following constants are defined to describe the operating system's file system:

```

LineEnding = #10;
LFNSupport = true;
DirectorySeparator = '/';
DriveSeparator = ':';
PathSeparator = ':';
FileNameCaseSensitive : Boolean = True;

```

The shown values are for unix platforms, but will be different on other platforms. The meaning of the constants is the following:

LineEnding End of line marker. This constant is used when writing end of lines to text files.

LFNSupport This is True if the system supports long file names, i.e. filenames that are not restricted to 8.3 characters.

DirectorySeparator The character that is used as a directory separator, i.e. it appears between various parts of a path to a file.

DriveSeparator On systems that support drive letters, this character separates the drive indication from the rest of a filename.

PathSeparator This character can be found between elements in a series of paths (such as the contents of the PATH environment variable).

FileNameCaseSensitive Indicates whether filenames are case sensitive.

When programming cross-platform, use these constants instead of hard-coded characters. This will enhance portability of an application.

28.9 Integer Types

The following integer types are defined in the System unit:

```
Shortint = -128..127;
SmallInt = -32768..32767;
Longint = $80000000..$7fffffff;
byte = 0..255;
word = 0..65535;
dword = longword;
cardinal = longword;
Integer = smallint;
```

The following types are used for the functions that need compiler magic such as Val ([1054](#)) or Str ([1046](#)):

```
StrLenInt = LongInt;
ValsInt = Longint;
ValUInt = Cardinal;
ValReal = Extended;
```

The Real48 type is defined to emulate the old Turbo Pascal Real type:

```
Real48 = Array[0..5] of byte;
```

The assignment operator has been overloaded so this type can be assigned to the Free Pascal native Double and Extended types. Real2Double ([1031](#)).

The following character types are defined for Delphi compatibility:

```
TAnsiChar = Char;
AnsiChar = TAnsiChar;
```

And the following pointer types as well:

```
PChar = ^char;
pPChar = ^PChar;
PAnsiChar = PChar;
PQWord = ^QWord;
```

```

PInt64      = ^Int64;
pshortstring = ^shortstring;
plongstring  = ^longstring;
pansistring  = ^ansistring;
pwidestring  = ^widestring;
pextended    = ^extended;
ppointer     = ^pointer;

```

For the SetJmp (1039) and LongJmp (1019) calls, the following jump bufer type is defined (for the I386 processor):

```

jmp_buf = record
    ebx,esi,edi : Longint;
    bp,sp,pc : Pointer;
end;
PJmp_buf = ^jmp_buf;

```

The following records and pointers can be used to scan the entries in the string message handler tables:

```

tmsgstrtable = record
    name : pshortstring;
    method : pointer;
end;
pmsgstrtable = ^tmsgstrtable;

tstringmessagetable = record
    count : dword;
    msgstrtable : array[0..0] of tmsgstrtable;
end;
pstringmessagetable = ^tstringmessagetable;

```

The base class for all classes is defined as:

```

Type
TObject = Class
Public
    constructor create;
    destructor destroy;virtual;
    class function newinstance : tobject;virtual;
    procedure freeinstance;virtual;
    function safecallexception(exceptobject : tobject;
        exceptaddr : pointer) : longint;virtual;
    procedure defaulthandler(var message);virtual;
    procedure free;
    class function initinstance(instance : pointer) : tobject;
    procedure cleanupinstance;
    function classtype : tclass;
    class function classinfo : pointer;
    class function classname : shortstring;
    class function classnameis(const name : string) : boolean;
    class function classparent : tclass;
    class function instancesize : longint;

```

```

class function inheritsfrom(aclass : tclass) : boolean;
class function inheritsfrom(aclass : tclass) : boolean;
class function stringmessagetable : pstringmessagetable;
procedure dispatch(var message);
procedure dispatchstr(var message);
class function methodaddress(const name : shortstring) : pointer;
class function methodname(address : pointer) : shortstring;
function fieldaddress(const name : shortstring) : pointer;
procedure AfterConstruction;virtual;
procedure BeforeDestruction;virtual;
procedure DefaultHandlerStr(var message);virtual;
end;
TClass = Class Of TObject;
PClass = ^TClass;

```

Unhandled exceptions can be treated using a constant of the TExceptProc type:

```
TExceptProc = Procedure (Obj : TObject; Addr,Frame: Pointer);
```

Obj is the exception object that was used to raise the exception, Addr and Frame contain the exact address and stack frame where the exception was raised.

The TVarRec type is used to access the elements passed in a \var{Array of Const} argument to a function or procedure:

Type

```

PVarRec = ^TVarRec;
TVarRec = record
  case VType : Longint of
    vtInteger      : (VInteger: Longint);
    vtBoolean      : (VBoolean: Boolean);
    vtChar         : (VChar: Char);
    vtExtended     : (VExtended: PExtended);
    vtString       : (VString: PShortString);
    vtPointer      : (VPointer: Pointer);
    vtPChar        : (VPChar: PChar);
    vtObject       : (VObject: TObject);
    vtClass        : (VClass: TClass);
    vtAnsiString   : (VAnsiString: Pointer);
    vtWideString   : (VWideString: Pointer);
    vtInt64        : (VInt64: PInt64);
  end;

```

The heap manager uses the TMemoryManager type:

```

PMemoryManager = ^TMemoryManager;
TMemoryManager = record
  Getmem      : Function(Size:Longint):Pointer;
  Freemem     : Function(var p:pointer):Longint;
  FreememSize : Function(var p:pointer;Size:Longint):Longint;
  AllocMem    : Function(Size:longint):Pointer;
  ReAllocMem  : Function(var p:pointer;Size:longint):Pointer;
  MemSize     : function(p:pointer):Longint;
  MemAvail    : Function:Longint;

```

```

    MaxAvail      : Function:Longint;
    HeapSize      : Function:Longint;
end;
```

More information on using this record can be found in `\progref`.

28.10 Overview

The system unit contains the standard supported functions of Free Pascal. It is the same for all platforms. Basically it is the same as the system unit provided with Borland or Turbo Pascal.

Functions are listed in alphabetical order. Arguments of functions or procedures that are optional are put between square brackets.

The pre-defined constants and variables are listed in the first section. The second section contains an overview of all functions, grouped by functionality, and the last section contains the supported functions and procedures.

28.11 Constants, types and variables

Constants

```
AbstractErrorProc : TAbstractErrorProc = nil
```

If set, the `AbstractErrorProc` constant is used when an abstract error occurs. If it is not set, then the standard error handling is done: A stack dump is performed, and the program exits with error code 211.

The `SysUtils` unit sets this procedure and raises an exception in its handler.

```
AssertErrorProc : TAssertErrorProc = @SysAssert
```

If set, the `AbstractErrorProc` constant is used when an assert error occurs. If it is not set, then the standard error handling is done: The assertion error message is printed, together with the location of the assertion, and A stack dump is performed, and the program exits with error code 227.

The `SysUtils` unit sets this procedure and raises an exception in its handler.

```
BackTraceStrFunc : TBackTraceStrFunc = @SysBackTraceStr
```

This handler is called to get a standard format for the backtrace routine.

```
CmdLine : PChar = nil
```

Current command-line.

```
DefaultTextLineBreakStyle : TTextLineBreakStyle = tlbsLF
```

`DefaultTextLineBreakStyle` contains the default OS setting for the `TTextLineBreakStyle` (967) type. It is initialized by the system unit, and is used to determine the default line ending when writing to text files.

This constant is part of a set of constants that describe the OS characteristics. These constants should be used instead of hardcoding OS characteristics.

```
DirectorySeparator = '/'
```

`DirectorySeparator` is the character used by the current operating system to separate directory parts in a pathname. This constant is system dependent, and should not be set.

This constant is part of a set of constants that describe the OS characteristics. These constants should be used instead of hardcoding OS characteristics.

```
DriveSeparator = ':'
```

On systems that support driveletters, the `DriveSeparator` constant denotes the character that separates the drive indicator from the directory part in a filename path.

This constant is part of a set of constants that describe the OS characteristics. These constants should be used instead of hardcoding OS characteristics.

```
Erroraddr : pointer = nil
```

Address where the last error occurred.

```
Errorcode : Word = 0
```

Last error code.

```
ErrorProc : TErrorProc = nil
```

If set, the `ErrorProc` constant is used when a run-time error occurs. If it is not set, then the standard error handling is done: a stack dump is performed, and the program exits with the indicated error code.

The `SysUtils` unit sets this procedure and raises an exception in its handler.

```
ExceptProc : TExceptProc = nil
```

This constant points to the current exception handling procedure. This routine is called when an unhandled exception occurs, i.e. an exception that is not stopped by a `except` block.

If the handler is not set, the RTL will emit a run-time error 217 when an unhandler exception occurs.

It is set by the `sysutils` (1071) unit.

```
ExitProc : pointer = nil
```

Exit procedure pointer.

```
Filemode : Byte = 2
```

Default file mode for untyped files.

```
FileNameCaseSensitive : Boolean = true
```

`FileNameCaseSensitive` is `True` if case is important when using filenames on the current OS. In this case, the OS will treat files with different cased names as different files. Note that this may depend on the filesystem: Unix operating systems that access a DOS or Windows partition will have this constant set to `true`, but when writing to the DOS partition, the casing is ignored.

This constant is part of a set of constants that describe the OS characteristics. These constants should be used instead of hardcoding OS characteristics.

fmAppend = \$D7B4

File mode: File is open for writing, appending to the end.

fmClosed = \$D7B0

File mode: File is closed.

fmInOut = \$D7B3

File mode: File is open for reading and writing.

fmInput = \$D7B1

File mode: File is open for reading.

fmOutput = \$D7B2

File mode: File is open for writing.

fpc_in_abs_real = 122

FPC compiler internal procedure index: abs (real)

fpc_in_addr_x = 42

FPC compiler internal procedure index: addr

fpc_in_arctan_real = 125

FPC compiler internal procedure index: arctan (real)

fpc_in_assert_x_y = 41

FPC compiler internal procedure index: assert

fpc_in_assigned_x = 19

FPC compiler internal procedure index: assigned

fpc_in_break = 39

FPC compiler internal procedure index: break

fpc_in_chr_byte = 7

FPC compiler internal procedure index: chr

fpc_in_concat_x = 18

FPC compiler internal procedure index: concat

fpc_in_const_abs = 103

FPC compiler internal procedure index: abs

fpc_in_const_arctan = 112

FPC compiler internal procedure index: arctan

fpc_in_const_cos = 113

FPC compiler internal procedure index: cos

fpc_in_const_exp = 114

FPC compiler internal procedure index: exp

fpc_in_const_frac = 102

FPC compiler internal procedure index: frac

fpc_in_const_int = 104

FPC compiler internal procedure index: int

fpc_in_const_ln = 115

FPC compiler internal procedure index: in

fpc_in_const_odd = 106

FPC compiler internal procedure index: sqr

fpc_in_const_pi = 110

FPC compiler internal procedure index: pi

fpc_in_const_ptr = 107

FPC compiler internal procedure index: sqr

fpc_in_const_round = 101

FPC compiler internal procedure index: round

fpc_in_const_sin = 116

FPC compiler internal procedure index: sin

fpc_in_const_sqr = 105

FPC compiler internal procedure index: sqr

fpc_in_const_sqrt = 111

FPC compiler internal procedure index: sqrt

fpc_in_const_swap_long = 109

FPC compiler internal procedure index: swap (long)

`fpc_in_const_swap_qword` = 128

FPC compiler internal procedure index: swap (qword)

`fpc_in_const_swap_word` = 108

FPC compiler internal procedure index: swap (word)

`fpc_in_const_trunc` = 100

FPC compiler internal procedure index: trunc

`fpc_in_continue` = 40

FPC compiler internal procedure index: continue

`fpc_in_copy_x` = 49

FPC compiler internal procedure index: copy

`fpc_in_cos_real` = 119

FPC compiler internal procedure index: cos (real)

`fpc_in_cycle` = 52

FPC compiler internal procedure index: cycle

`fpc_in_dec_x` = 36

FPC compiler internal procedure index: dec

`fpc_in_dispose_x` = 47

FPC compiler internal procedure index: dispose

`fpc_in_exclude_x_y` = 38

FPC compiler internal procedure index: exclude

`fpc_in_exit` = 48

FPC compiler internal procedure index: exit

`fpc_in_finalize_x` = 45

FPC compiler internal procedure index: finalize

`fpc_in_high_x` = 28

FPC compiler internal procedure index: high

`fpc_in_hi_long` = 4

FPC compiler internal procedure index: hi (long)

`fpc_in_hi_qword` = 118

FPC compiler internal procedure index: hi (qword)

`fpc_in_hi_word` = 2

FPC compiler internal procedure index: hi (word)

`fpc_in_include_x_y` = 37

FPC compiler internal procedure index: include

`fpc_in_inc_x` = 35

FPC compiler internal procedure index: inc

`fpc_in_initialize_x` = 50

FPC compiler internal procedure index: initialize

`fpc_in_leave` = 51

FPC compiler internal procedure index: leave

`fpc_in_length_string` = 6

FPC compiler internal procedure index: length

`fpc_in_ln_real` = 126

FPC compiler internal procedure index: ln (real)

`fpc_in_low_x` = 27

FPC compiler internal procedure index: low

`fpc_in_lo_long` = 3

FPC compiler internal procedure index: lo (long)

`fpc_in_lo_qword` = 117

FPC compiler internal procedure index: lo (qword)

`fpc_in_lo_word` = 1

FPC compiler internal procedure index: lo (word)

`fpc_in_mmx_pcmpeqb` = 200

FPC compiler internal procedure index: MMX

`fpc_in_mmx_pcmpeqd` = 202

FPC compiler internal procedure index: MMX

`fpc_in_mmx_pcmpeqw` = 201

FPC compiler internal procedure index: MMX

`fpc_in_mmx_pcmpgtb` = 203

FPC compiler internal procedure index: MMX

`fpc_in_mmx_pcmpgtd` = 205

FPC compiler internal procedure index: MMX

`fpc_in_mmx_pcmpgtw` = 204

FPC compiler internal procedure index: MMX

`fpc_in_new_x` = 46

FPC compiler internal procedure index: new

`fpc_in_ofs_x` = 21

FPC compiler internal procedure index: ofs

`fpc_in_ord_x` = 5

FPC compiler internal procedure index: ord

`fpc_in_pi` = 121

FPC compiler internal procedure index: pi

`fpc_in_pred_x` = 30

FPC compiler internal procedure index: pred

`fpc_in_prefetch_var` = 129

FPC compiler internal procedure index: prefetch

`fpc_in_readln_x` = 17

FPC compiler internal procedure index: readln

`fpc_in_read_x` = 16

FPC compiler internal procedure index: read

`fpc_in_reset_typedfile` = 32

FPC compiler internal procedure index: reset

`fpc_in_reset_x` = 25

FPC compiler internal procedure index: reset

`fpc_in_rewrite_typedfile` = 33

FPC compiler internal procedure index: rewrite

`fpc_in_rewrite_x` = 26

FPC compiler internal procedure index: rewrite

`fpc_in_seg_x` = 29

FPC compiler internal procedure index: seg

`fpc_in_setlength_x` = 44

FPC compiler internal procedure index: setlength

`fpc_in_settextbuf_file_x` = 34

FPC compiler internal procedure index: settextbuf

`fpc_in_sin_real` = 127

FPC compiler internal procedure index: sin (real)

`fpc_in_sizeof_x` = 22

FPC compiler internal procedure index: sizeof

`fpc_in_sqrt_real` = 124

FPC compiler internal procedure index: sqrt (real)

`fpc_in_sqr_real` = 123

FPC compiler internal procedure index: sqr (real)

`fpc_in_str_x_string` = 20

FPC compiler internal procedure index: str

`fpc_in_succ_x` = 31

FPC compiler internal procedure index: succ

`fpc_in_typeinfo_x` = 43

FPC compiler internal procedure index: typeinfo

`fpc_in_typeof_x` = 23

FPC compiler internal procedure index: typeof

`fpc_in_val_x` = 24

FPC compiler internal procedure index: val

```
fpc_in_writeln_x = 15
```

FPC compiler internal procedure index: writeln

```
fpc_in_write_x = 14
```

FPC compiler internal procedure index: write

```
growheapsize1 : PtrInt = 256 * 1024
```

Grow rate for block less than 256 Kb.

```
growheapsize2 : PtrInt = 1024 * 1024
```

Grow rate for block larger than 256 Kb.

```
growheapsize_small : PtrInt = 32 * 1024
```

Fixed size small blocks grow rate

```
IsMultiThread : Boolean = false
```

Indicates whether more than one thread is running in the application.

```
LFNSupport = true
```

LFNSupport determines whether the current OS supports long file names, i.e. filenames that are not of the form 8.3 as on ancient DOS systems. If the value of this constant is `True` then long filenames are supported. If it is false, then not.

This constant is part of a set of constants that describe the OS characteristics. These constants should be used instead of hardcoding OS characteristics.

```
LineEnding = #10
```

LineEnding is a constant which contains the current line-ending character. This character is system dependent, and is initialized by the system. It should not be set.

This constant is part of a set of constants that describe the OS characteristics. These constants should be used instead of hardcoding OS characteristics.

```
maxExitCode = 255
```

maxExitCode is the maximum value for the `Halt` (1007) call.

```
maxint = maxsmallint
```

Maximum integer value.

```
maxLongint = $7fffffff
```

Maximum longint value.

`MaxSIntValue = High (ValSInt)`

Maximum String-size value.

`maxSmallint = 32767`

Maximum smallint value.

`MaxUIntValue = High (ValUInt)`

Maximum unsigned integer value.

`Max_Frame_Dump : Word = 8`

Maximum number of frames to show in error frame dump.

`PathSeparator = ':'`

`PathSeparator` is the character used commonly on the current operating system to separate paths in a list of paths, such as the `PATH` environment variable.

This constant is part of a set of constants that describe the OS characteristics. These constants should be used instead of hardcoding OS characteristics.

`RaiseProc : TExceptProc = nil`

Procedure to raise an exception.

`SIGSTKSZ = 40960`

`sLineBreak = LineEnding`

`sLineBreak` is an alias for `LineEnding` ([951](#)) and is supplied for Delphi compatibility.

This constant is part of a set of constants that describe the OS characteristics. These constants should be used instead of hardcoding OS characteristics.

`StackError : Boolean = false`

Indicate whether there was a stack error.

`StdErrorHandle = 2`

Value of the OS handle for the standard error-output file.

`StdInputHandle = 0`

Value of the OS handle for the standard input file.

`StdOutputHandle = 1`

Value of the OS handle for the standard output file.

`UnusedHandle = -1`

Value indicating an unused file handle (as reported by the OS).

`varany = $101`

Variant type: Any

`vararray = $2000`

Variant type: variant Array

`varboolean = 11`

Variant type: Boolean type

`varbyref = $4000`

Variant type: By reference

`varbyte = 17`

Variant type: Byte (8 bit)

`varcurrency = 6`

Variant type: Currency

`vardate = 7`

Variant type: Date

`vardecimal = 14`

Variant type: Decimal (BCD)

`vardispatch = 9`

Variant type: dispatch interface

`vardouble = 5`

Variant type: Double float

`vareempty = 0`

Variant type: Empty variant

`varerror = 10`

Variant type: Error type

`varint64 = 20`

Variant type: Integer (64-Bit)

`varinteger = 3`

Variant type: Integer (32-bit)

`varlongword = 19`

Variant type: Word (32 bit)

`varnull = 1`

Variant type: Null (971) variant

`varolestr = 8`

Variant type: OLE string (wdestring)

`varqword = 21`

Variant type: Word (64-bit)

`varshortint = 16`

Variant type: Shortint (16 bit)

`varsingle = 4`

Variant type: Single float

`varsmallint = 2`

Variant type: smallint (8 bit)

`varstrarg = $48`

Variant type: String

`varstring = $100`

Variant type: String

`vartypemask = $fff`

Variant type: Mask to extract type

`varunknown = 13`

Variant type: Unknown

`varvariant = 12`

Variant type: Variant (arrays only)

`varword = 18`

Variant type: Word (16 bit)

`varword64 = varqword`

Variant type: Word (64-bit)

`vmtAfterConstruction = vmtMethodStart + sizeof (pointer) * 5`

VMt Layout: ?

`vmtAutoTable = vmtParent + sizeof (pointer) * 7`

VMt layout: ?

`vmtBeforeDestruction = vmtMethodStart + sizeof (pointer) * 6`

VMt Layout: ?

`vmtClassName = vmtParent + sizeof (pointer)`

VMt Layout: location of class name.

`vmtDefaultHandler = vmtMethodStart + sizeof (pointer) * 4`

VMt Layout: ?

`vmtDefaultHandlerStr = vmtMethodStart + sizeof (pointer) * 7`

VMt Layout: ?

`vmtDestroy = vmtMethodStart`

VMt Layout: Location of destructor pointer.

`vmtDynamicTable = vmtParent + sizeof (pointer) * 2`

VMt Layout: location of dynamic methods table.

`vmtFieldTable = vmtParent + sizeof (pointer) * 4`

VMt Layout: Location of fields table.

`vmtFreeInstance = vmtMethodStart + sizeof (pointer) * 2`

VMt Layout: location of FreeInstance method.

`vmtInitTable = vmtParent + sizeof (pointer) * 6`

VMt Layout: ?

`vmtInstanceSize = 0`

VMt Layout: Location of class instance size in VMT

`vmtIntfTable = vmtParent + sizeof (pointer) * 8`

VMt layout: Interface table

`vmtMethodStart = vmtParent + sizeof (pointer) * 10`

VMT layout: start of method table.

```
vmtMethodTable = vmtParent + sizeof ( pointer ) * 3
```

VMT Layout: Method table start.

```
vmtMsgStrPtr = vmtParent + sizeof ( pointer ) * 9
```

VMT layout: message strings table.

```
vmtNewInstance = vmtMethodStart + sizeof ( pointer )
```

VMT Layout: location of NewInstance method.

```
vmtParent = sizeof ( ptring ) * 2
```

VMT Layout: location of pointer to parent VMT.

```
vmtSafeCallException = vmtMethodStart + sizeof ( pointer ) * 3
```

VMT Layout: ?

```
vmtTypeInfo = vmtParent + sizeof ( pointer ) * 5
```

VMT Layout: Location of class type information.

```
vtAnsiString = 11
```

TVarRec type: Ansistring

```
vtBoolean = 1
```

TVarRec type: Boolean

```
vtChar = 2
```

TVarRec type: Char

```
vtClass = 8
```

TVarRec type: Class type

```
vtCurrency = 12
```

TVarRec type: Currency

```
vtExtended = 3
```

TVarRec type: Extended

```
vtInt64 = 16
```

TVarRec type: Int64 (signed 64-bit integer)

```
vtInteger = 0
```

TVarRec type: Integer

vtInterface = 14

TVarRec type: Interface

vtObject = 7

TVarRec type: Object instance

vtPChar = 6

TVarRec type: PChar

vtPointer = 5

TVarRec type: pointer

vtPWideChar = 10

TVarRec type: PWideChar

vtQWord = 17

TVarRec type: QWord (unsigned 64-bit integer)

vtString = 4

TVarRec type: String

vtVariant = 13

TVarRec type: Variant

vtWideChar = 9

TVarRec type: Widechar

vtWideString = 15

TVarRec type: WideString

Types

AnsiChar = Char

Alias for 1-byte sized char.

Byte = 0..255

An unsigned 8-bits integer

Cardinal = LongWord

An unsigned 32-bits integer.

Currency = Int64

Currency type.

DWord = LongWord

An unsigned 32-bits integer

Error = LongInt

32-bit signed integer.

fpc_big_chararray = Array[0..1023] of Char

Array of char.

fpc_normal_set = Array[0..7] of LongInt

Type with the size of a normal set

fpc_small_set = LongInt

Type with the size of a small set

HRESULT = LongInt

32-Bit signed integer.

Integer = SmallInt

The system unit defines Integer as a signed 16-bit integer. But when DELPHI or OBJFPC mode are active, then the **objpas** unit redefines Integer as a 16-bit integer.

IntegerArray = Array[0..\$effffff] of Integer

Generic array of integer.

```
jmp_buf = packed record
  ebx : LongInt;
  esi : LongInt;
  edi : LongInt;
  bp  : Pointer;
  sp  : Pointer;
  pc  : Pointer;
end
```

Record type to store processor information.

Longint = + (- 2147483647 - 1) .. \$7fffffff

A signed 32-bits integer

`PAnsiChar = PChar`

Alias for PChar (959) type.

`PAnsiString = ^AnsiString`

Pointer to an ansistring type.

`PBoolean = ^Boolean`

Pointer to a Boolean type.

`PByte = ^Byte`

Pointer to byte (957) type

`pcallldesc = ^tcallldesc`

Pointer to TCallDesc (964) record.

`PCardinal = ^Cardinal`

Pointer to Cardinal (958) type

`PChar = ^Char`

Or the same as a pointer to an array of char. See the reference manual for more information about this type.

`PClass = ^TClass`

Pointer to TClass (964)

`PCurrency = ^Currency`

Pointer to currency type.

`PDate = ^TDateTime`

Pointer to a TDateTime (964) type.

`pdispdesc = ^tdispdesc`

Pointer to tdispdesc (964) record

`PDouble = ^Double`

Pointer to double-sized float value.

`PDWord = ^DWord`

Pointer to DWord (958) type

`pdynarrayindex = ^tdynarrayindex`

Pointer to `tdynarrayindex` (964) type.

`PError` = `^Error`

Pointer to an `Error` (958) type.

`PExceptObject` = `^TExceptObject`

Pointer to Exception handler procedural type `TExceptProc` (965)

`PExtended` = `^Extended`

Pointer to extended-sized float value.

`PGuid` = `^TGuid`

Pointer to `TGUID` (965) type.

`PInt64` = `^Int64`

Pointer to `Int64` type

`PInteger` = `^LongInt`

Pointer to integer (958) type

`PIntegerArray` = `^IntegerArray`

Pointer to `IntegerArray` (958) type

`pinterfaceentry` = `^tinterfaceentry`

Pointer to `tinterfaceentry` (965) record.

`pinterfacetable` = `^tinterfacetable`

Pointer to `tinterfacetable` (966) record.

`PJump_buf` = `^jmp_buf`

Pointer to `jmp_buf` (958) record

`PLongBool` = `^LongBool`

Pointer to a `LongBool` type.

`PLongint` = `^LongInt`

Pointer to `Longint` (958) type

`PLongWord` = `^LongWord`

Pointer to `LongWord` type

`PMemoryManager` = `^TMemoryManager`

Pointer to TMemoryManager (966) record

PMsgStrTable = ^TMsgStrTable

Pointer to array of TMsgStrTable (966) records.

PointerArray = Array[0..512*1024*1024-2] of Pointer

Generic pointer array.

POleVariant = ^OleVariant

Pointer to OleVariant type.

PPAnsiChar = PPChar

Alias for PPChar (961) type.

PPChar = ^PChar

Pointer to an array of pointers to null-terminated strings.

PPCharArray = ^TPCharArray

Pointer to TPCharArray (966) type.

PPointer = ^Pointer

Pointer to a pointer type.

PPointerArray = ^PointerArray

Pointer to PointerArray (961) type

PPPointer = ^PPPointer

Pointer to a PPointer (961) type.

PPtrInt = ^PtrInt

Pointer to PtrInt (962) type.

PPWideChar = ^PWideChar

Pointer to link id="PWideChar"> type.

PQWord = ^QWord

Pointer to QWord type

PShortInt = ^ShortInt

Pointer to shortint (963) type

PShortString = ^ShortString

Pointer to a shortstring type.

`PSingle = ^Single`

Pointer to single-sized float value.

`PSizeInt = ^SizeInt`

Pointer to a `SizeInt` (963) type

`PSmallInt = ^SmallInt`

Pointer to `smallint` (963) type

`pstringmessagetable = ^TStringMessageTable`

Pointer to `TStringMessageTable` (967) record.

`PText = ^Text`

Pointer to text file.

`PtrInt = LongInt`

`PtrInt` is an integer type which has always the same size as a pointer. When using integers which will be cast to pointers and vice versa, use this type, never the regular integer type.

`PtrUInt = DWord`

`PtrUInt` is an unsigned integer type which has always the same size as a pointer. When using integers which will be cast to pointers and vice versa, use this type, never the regular `Cardinal` type.

`PUCS2Char = PWideChar`

Pointer to `UCS2Char` (969) character.

`PUCS4Char = ^UCS4Char`

Pointer to `UCS4Char` (970)

`PUCS4CharArray = ^TUCS4CharArray`

Pointer to array of `UCS4Char` (970) characters.

`PUTF8String = ^UTF8String`

Pointer to `UTF8String` (970)

`pvararray = ^tvararray`

Pointer to `TVarArray` (967) type.

`pvararraybound = ^tvararraybound`

Pointer to `tvararraybound` (967) type.

`pvararrayboundarray = ^tvararrayboundarray`

Pointer to `tvararrayboundarray` (967) type.

`pvararraycoorarray = ^tvararraycoorarray`

Pointer to `tvararraycoorarray` (967) type.

`pvardata = ^tvardata`

Pointer to `TVarData` (968) record.

`PVariant = ^Variant`

Pointer to `Variant` type.

`pvariantmanager = ^tvariantmanager`

Pointer to `TVariantManager` (968) record.

`PVarRec = ^TVarRec`

Pointer to `TVarRec` (969) type.

`PWideChar = ^WideChar`

Pointer to `WChar` (970).

`PWord = ^Word`

Pointer to `word` (970) type

`PWordBool = ^WordBool`

Pointer to a `WordBool` type.

`real48 = Array[0..5] of Byte`

TP compatible real type (6 bytes) definition

`ShortInt = -128..127`

A signed 8-bits integer

`SizeInt = LongInt`

Signed integer type which fits for sizes

`SizeUInt = DWord`

Unsigned Integer type which fits for sizes

`SmallInt = -32768..32767`

A signed 16-bits integer

`TAbstractErrorProc` = procedure

Abstract error handler procedural type.

`TAnsiChar` = Char

Alias for 1-byte sized char.

```
TAssertErrorProc = procedure(const msg: ShortString;
                             const fname: ShortString; lineno: LongInt;
                             erroraddr: pointer)
```

Assert error handler procedural type.

`TBackTraceStrFunc` = function(Addr: Pointer) : ShortString

Type for formatting of backtrace dump.

`TBoundArray` = Array[] of Integer

Dynamic array of integer.

```
tcalldesc = packed record
  calltype : Byte;
  argcount : Byte;
  namedargcount : Byte;
  argtypes : Array[0..255] of Byte;
end
```

`tcalldesc` is used to encode the arguments to a dispatch call to an OLE dual interface. It is used on windows only. It describes the arguments to a call.

`TClass` = Class of TObject

Class of TObject ([1055](#)).

`TDateTime` = Double

Encoded Date-Time type.

```
tdispdesc = packed record
  dispid : LongInt;
  restype : Byte;
  calldesc : tcalldesc;
end
```

`tcalldesc` is used to encode a dispatch call to an OLE dispatch interface. It is used on windows only. It describes the dispatch call.

`tdynarrayindex` = SizeInt

A variable of type `tdynarrayindex` will always have the correct size, suitable for serving as an index in a dynamic array.

```
TErrorProc = procedure(ErrNo: LongInt;Address: Pointer;Frame: Pointer)
```

Standard error handler procedural type.

```
TExceptObject = record
  FObject : TObject;
  Addr : pointer;
  Frame : pointer;
  Next : PExceptObject;
  refcount : LongInt;
end
```

TExceptObject is the exception description record which is found on the exception stack.

```
TExceptProc = procedure(Obj: TObject;Addr: Pointer;Frame: Pointer)
```

Exception handler procedural type

```
TextFile = Text
```

Alias for Text file type.

```
TGuid = packed record
end
```

Standard GUID representation type.

```
THandle = LongInt
```

This type should be considered opaque. It is used to describe file and other handles.

```
THeapStatus = record
  MaxHeapSize : PtrInt;
  MaxHeapUsed : PtrInt;
  CurrHeapSize : PtrInt;
  CurrHeapUsed : PtrInt;
  CurrHeapFree : PtrInt;
end
```

THeapStatus is the record describing the current heap status. It is returned by the GetHeapStatus (1005) call.

```
tinterfaceentry = packed record
  IID : PGuid;
  VTable : Pointer;
  IOffset : DWord;
  IIDStr : PShortString;
end
```

tinterfaceentry is used to store the list of Interfaces of a class. This list is stored as an array of tinterfaceentry records.

```

tinterfacetable = packed record
  EntryCount : Word;
  Entries : Array[0..0] of tinterfaceentry;
end

```

Record to store list of interfaces of a class.

```

TMemoryManager = record
  NeedLock : Boolean;
  Getmem : function(Size: PtrInt) : Pointer;
  Freemem : function(p: pointer) : PtrInt;
  FreememSize : function(p: pointer;Size: PtrInt) : PtrInt;
  AllocMem : function(Size: PtrInt) : Pointer;
  ReAllocMem : function(var p: pointer;Size: PtrInt) : Pointer;
  MemSize : function(p: pointer) : PtrInt;
  GetHeapStatus : procedure(var status: THeapStatus);
end

```

TMemoryManager describes the memory manager. For more information about the memory manager, see the programmer's reference.

```

TMemoryMutexManager = record
  MutexInit : procedure;
  MutexDone : procedure;
  MutexLock : procedure;
  MutexUnlock : procedure;
end

```

When the heapmanager needs a lock, then the mutex manager is used to handle the lock.

```

TMsgStrTable = record
  name : PShortString;
  method : pointer;
end

```

Record used in string message handler table.

```

TPCharArray = packed Array[0..(MaxLongintdivSizeOf(PChar))-1] of PChar

```

Array of PChar

```

TProcedure = procedure

```

Simple procedural type.

```

TStringMessageTable = record
  count : DWord;
  msgstrtable : Array[0..0] of TMsgStrTable;
end

```

Record used to describe the string messages handled by a class. It consists of a count, followed by an array of TMsgStrTable (966) records.

TTextLineBreakStyle = (tlbsLF,tlbsCRLF,tlbsCR)

Table 28.7: Enumeration values for type TTextLineBreakStyle

Value	Explanation
tlbsCR	Carriage-return (#13, Mac-OS style)
tlbsCRLF	Carriage-return, line-feed (#13#30, Windows style)
tlbsLF	Line-feed only (#10, unix style)

Text line break style. (end of line character)

TUCS4CharArray = Array[0..\$effffff] of UCS4Char

Array of UCS4Char (970) characters.

```
tvararray = packed record
  dimcount : Word;
  flags : Word;
  elementsize : LongInt;
  lockcount : LongInt;
  data : pointer;
  bounds : Array[0..255] of tvararraybound;
end
```

tvararray is a record describing a variant array. It contains some general data, followed by a number of TVarArrayBound (967) records equal to the number of dimensions in the array (dimcount).

```
tvararraybound = packed record
  elementcount : LongInt;
  lowbound : LongInt;
end
```

tvararraybound is used to describe one dimension in a variant array.

tvararrayboundarray = Array[0..0] of tvararraybound

array of tvararraybound (967) records.

tvararraycoorarray = Array[0..0] of LongInt

Array of variant array coordinates

```
tvardata = packed record
  vtype : tvartype;
end
```

TVarData is a record representation of a variant. It contains the internal structure of a variant and is handled by the various variant handling routines.

```

tvariantmanager = record
  vartoint : function(const v: variant) : LongInt;
  vartoint64 : function(const v: variant) : Int64;
  vartoword64 : function(const v: variant) : qword;
  vartobool : function(const v: variant) : Boolean;
  vartoreal : function(const v: variant) : extended;
  vartocurr : function(const v: variant) : Currency;
  vartopstr : procedure(var s; const v: variant);
  vartolstr : procedure(var s: ansistring; const v: variant);
  vartowstr : procedure(var s: widestring; const v: variant);
  vartointf : procedure(var intf: iinterface; const v: variant);
  vartodisp : procedure(var disp: idispatch; const v: variant);
  vartodynarray : procedure(var dynarr: pointer; const v: variant; typeinfo: pointer);
  varfrombool : procedure(var dest: variant; const source: Boolean);
  varfromint : procedure(var dest: variant; const source: LongInt; const Range: LongInt);
  varfromint64 : procedure(var dest: variant; const source: Int64);
  varfromword64 : procedure(var dest: variant; const source: qword);
  varfromreal : procedure(var dest: variant; const source: extended);
  varfrompstr : procedure(var dest: variant; const source: ShortString);
  varfromlstr : procedure(var dest: variant; const source: ansistring);
  varfromwstr : procedure(var dest: variant; const source: WideString);
  varfromintf : procedure(var dest: variant; const source: iinterface);
  varfromdisp : procedure(var dest: variant; const source: idispatch);
  varfromdynarray : procedure(var dest: variant; const source: pointer; typeinfo: pointer);
  olevarfrompstr : procedure(var dest: olevariant; const source: shortstring);
  olevarfromlstr : procedure(var dest: olevariant; const source: ansistring);
  olevarfromvar : procedure(var dest: olevariant; const source: variant);
  olevarfromint : procedure(var dest: olevariant; const source: LongInt;
    const range: ShortInt);
  varop : procedure(var left: variant; const right: variant; opcode: tvarop);
  cmpop : function(const left: variant; const right: variant; const opcode: tvarop)
    : Boolean;
  varneg : procedure(var v: variant);
  varnot : procedure(var v: variant);
  varinit : procedure(var v: variant);
  varclear : procedure(var v: variant);
  varaddref : procedure(var v: variant);
  varcopy : procedure(var dest: variant; const source: variant);
  varcast : procedure(var dest: variant; const source: variant; vartype: LongInt);
  varcastole : procedure(var dest: variant; const source: variant; vartype: LongInt);
  dispinvoke : procedure(dest: pvardata; const source: tvardata; calldesc: pcalldesc;
    params: pointer);
  vararrayredim : procedure(var a: variant; highbound: SizeInt);
  vararrayget : function(var a: variant; indexcount: SizeInt; indices: SizeInt) : variant;
  vararrayput : procedure(var a: variant; const value: variant; indexcount: SizeInt;
    indices: SizeInt);
  writevariant : function(var t: text; const v: variant; width: LongInt) : Pointer;
  write0Variant : function(var t: text; const v: Variant) : Pointer;
end

```

TVariantManager describes the variant manager as expected by the SetVariantManager (1042) call.

```
tvarop = (opadd,opsubtract,opmultiply,opdivide,opintdivide,opmodulus,
          opshiftright,opshiftright,opand,opor,opxor,opcompare,opnegate,
          opnot,opcmppeq,opcmpne,opcmlt,opcmlt,opcmlt,opcmlt,opcmlt)
```

Table 28.8: Enumeration values for type tvarop

Value	Explanation
opadd	Variant operation: Addition.
opand	Variant operation: Binary AND operation
opcmppeq	Variant operation: Compare equal.
opcmpge	Variant operation: Compare larger than or equal
opcmlt	Variant operation: Compare larger than
opcmlt	Variant operation: Compare less than or equal to
opcmlt	Variant operation: Compare less than.
opcmlt	Variant operation: Compare not equal
opcompare	Variant operation: Compare
opdivide	Variant operation: division
opintdivide	Variant operation: integer divide
opmodulus	Variant operation: Modulus
opmultiply	Variant operation: multiplication
opnegate	Variant operation: negation.
opnot	Variant operation: Binary NOT operation.
opor	Variant operation: Binary OR operation
opshiftright	Variant operation: Shift left
opshiftright	Variant operation: Shift right
opsubtract	Variant operation: Subtraction
opxor	Variant operation: binary XOR operation.

tvarop describes a variant operation. It is mainly used for the variant manager to implement the various conversions and mathematical operations on a variant.

```
TVarRec = record
end
```

TVarRec is a record generated by the compiler for each element in a array of const call. The procedure that receives the constant array receives an array of TVarRec elements, with lower bound zero and high bound equal to the number of elements in the array minus one (as returned by High(Args))

```
tvarotype = Word
```

Type with size of variant type.

```
UCS2Char = WideChar
```

UCS2 unicode character.

```
UCS4Char = LongWord
```

UCS unicode character (unsigned 32 bit word)

UCS4String = Array[] of UCS4Char

String of UCS4Char (970) characters.

UTF8String = ansistring

UTF-8 unicode (ansi) string.

ValSInt = LongInt

Integer with the same size as the return code of the Val (1054) function.

ValUInt = Cardinal

Integer with the same size as the return code of the Val (1054) function.

WChar = Widechar

Wide char (16-bit sized char)

Word = 0..65535

An unsigned 16-bits integer

Variables

argc : LongInt; external name operatingsystem_parameter_argc

argc contains the number of command-line arguments passed to the program by the OS. It is not available on all systems.

argv : PPChar; external name operatingsystem_parameter_argv

argv contains a pointer to a nil-terminated array of null-terminated strings, containing the command-line arguments passed to the program by the OS. It is not available on all systems.

DispCallByIDProc : pointer

VarDispProc is called by the compiler if it needs to perform an interface call from a variant which contains a dispatch interface. For instance, the following call:

```
Var
  V : OleVariant;
begin
  (V as IWord).OpenDocument('c:\temp\mydoc.doc');
end;
```

where IWord is a dispatch interface is encoded by the compiler and passed to DispCallByIDProc.

This pointer must be set by a routine that calls the OS COM handling routines.

envp : PPChar; external name operatingsystem_parameter_envp

`envp` contains a pointer to a nil-terminated array of null-terminated strings, containing the environment variables passed to the program by the OS. It is not available on all systems.

`ErrOutput` : Text

`ErrOutput` is provided for Delphi compatibility.

`ExitCode` : Word;public name `operatingsystem_result`

Exit code for the program, will be communicated to the OS on exit.

`fpc_threadvar_relocate_proc` : pointer;public name `FPC_THREADVAR_RELOCATE`

`InOutRes` : Word

Result of last I/O operation. Read-Only.

`Input` : Text

Standard input text file.

`IsConsole` : Boolean

True for console applications, False for GUI applications.

`IsLibrary` : Boolean

True if the current module is a library. Otherwise module is an executable

`Null` : Variant

Null variant

`Output` : Text

Standard output text file.

`RandSeed` : Cardinal

Seed for Random ([1029](#)) function.

`ReturnNilIfGrowHeapFails` : Boolean

`ReturnNilIfGrowHeapFails` describes what happens if there is no more memory available from the operating system. if set to `True` the memory manager will return `Nil`. If set to `False` then a run-time error will occur.

`StackBottom` : Pointer

Current stack bottom.

`StackLength` : Cardinal

Maximum stack length.

StdErr : Text

Standard diagnostic output text file.

StdOut : Text

Alias for Output ([971](#)).

ThreadID : SizeUInt

Current Thread ID.

Unassigned : Variant

Unassigned variant.

VarDispProc : pointer

VarDispProc is called by the compiler if it needs to perform an interface call from a variant. For instance, the following call:

```
Var
  V : OleVariant;
begin
  V.OpenDocument('c:\temp\mydoc.doc');
end;
```

is encoded by the compiler and passed to VarDispProc.

This pointer must be set by a routine that calls the OS COM handling routines.

28.12 Procedures and functions

abs

Synopsis: Calculate absolute value

Declaration: `function abs(l: LongInt) : LongInt`
`function abs(l: Int64) : Int64`
`function abs(d: ValReal) : ValReal`

Visibility: default

Description: Abs returns the absolute value of a variable. The result of the function has the same type as its argument, which can be any numerical type.

Errors: None.

See also: Round ([1035](#))

Listing: ./refex/ex1.pp

Program Example1;

{ Program to demonstrate the Abs function. }

```

Var
  r : real;
  i : integer;

begin
  r:=abs(-1.0);    { r:=1.0 }
  i:=abs(-21);     { i:=21 }
end.

```

AbstractError

Synopsis: Generate an abstract error.

Declaration: `procedure AbstractError`

Visibility: default

Description: `AbstractError` generates an abstract error (run-time error 211). If the `AbstractErrorProc` ([943](#)) constant is set, it will be called instead.

Errors: This routine causes a run-time error 211.

See also: `AbstractErrorProc` ([943](#))

AcquireExceptionObject

Synopsis: Obtain a reference to the current exception object

Declaration: `function AcquireExceptionObject : Pointer`

Visibility: default

Description: `AcquireExceptionObject` returns the current exception object. It raises the reference count of the exception object, so it will not be freed. Calling this method is only valid within an except block.

The effect of this function is countered by re-raising an exception via `raise`;

To make sure that the exception object is released when it is no longer needed, `ReleaseExceptionObject` ([1032](#)) must be called when the reference is no longer needed.

Errors: If there is no current exception, a run-time error 231 will occur.

See also: `ReleaseExceptionObject` ([1032](#))

AddExitProc

Synopsis: Add an exit procedure to the exit procedure chain.

Declaration: `procedure AddExitProc(Proc: TProcedure)`

Visibility: default

Description: `AddExitProc` adds `Proc` to the exit procedure chain. At program exit, all procedures added in this way will be called in reverse order.

Errors: None.

See also: `ExitProc` ([944](#))

Addr

Synopsis: Return address of a variable

Declaration: `function Addr(X: TAnytype) : Pointer`

Visibility: default

Description: `Addr` returns a pointer to its argument, which can be any type, or a function or procedure name. The returned pointer isn't typed. The same result can be obtained by the `@` operator, which can return a typed pointer (`\progref`).

Errors: None

See also: `SizeOf` ([1043](#))

Listing: `./refex/ex2.pp`

Program `Example2`;

{ Program to demonstrate the Addr function. }

Const `Zero : integer = 0;`

Var `p : pointer;`
`i : Integer;`

begin

`p:=Addr(p);` *{ P points to itself }*

`p:=Addr(i);` *{ P points to i }*

`p:=Addr(Zero);` *{ P points to 'Zero' }*

end.

Align

Synopsis: Return aligned version of an address

Declaration: `function Align(Addr: PtrInt;Alignment: PtrInt) : PtrInt`
`function Align(Addr: Pointer;Alignment: PtrInt) : Pointer`

Visibility: default

Description: `Align` returns `Address`, aligned to `Alignment` bytes.

Errors: None.

AllocMem

Synopsis: Alias for `GetMem` ([1005](#))

Declaration: `function AllocMem(Size: PtrInt) : pointer`

Visibility: default

Description: `AllocMem` is an alias for `GetMem` ([1005](#)).

See also: `GetMem` ([1005](#))

Append

Synopsis: Open a file in append mode

Declaration: `procedure Append(var t: Text)`

Visibility: default

Description: Append opens an existing file in append mode. Any data written to F will be appended to the file. Only text files can be opened in append mode. After a call to Append, the file F becomes write-only. File sharing is not taken into account when calling Append.

Errors: If the file doesn't exist when appending, a run-time error will be generated. This behaviour has changed on Windows and Linux platforms, where in versions prior to 1.0.6, the file would be created in append mode.

See also: Rewrite ([1034](#)), Close ([982](#)), Reset ([1033](#))

Listing: ./refex/ex3.pp

Program Example3;

{ Program to demonstrate the Append function . }

Var f : text;

begin

Assign (f, 'test.txt');

Rewrite (f); *{ file is opened for write , and emptied }*

WriteIn (F, 'This is the first line of text.txt');

close (f);

Append(f); *{ file is opened for write , but NOT emptied.
any text written to it is appended. }*

WriteIn (f, 'This is the second line of text.txt');

close (f);

end.

arctan

Synopsis: Calculate inverse tangent

Declaration: `function arctan(d: ValReal) : ValReal`

Visibility: default

Description: Arctan returns the Arctangent of X, which can be any Real type. The resulting angle is in radial units.

Errors: None

See also: Sin ([1043](#)), Cos ([989](#))

Listing: ./refex/ex4.pp

Program Example4;

{ Program to demonstrate the ArcTan function . }

Var R : Real;

```

begin
  R:=ArcTan(0);      { R:=0 }
  R:=ArcTan(1)/pi;   { R:=0.25 }
end.

```

ArrayStringToPPchar

Synopsis: Convert an array of string to an array of null-terminated strings

Declaration: `function ArrayStringToPPchar(const S: Array[] of AnsiString;
reserveentries: LongInt) : PPChar`

Visibility: default

Description: `ArrayStringToPPchar` creates an array of null-terminated strings that point to strings which are the same as the strings in the array `S`. The function returns a pointer to this array. The array and the strings it contains must be disposed of after being used, because it they are allocated on the heap.

The `ReserveEntries` parameter tells `ArrayStringToPPchar` to allocate room at the end of the array for another `ReserveEntries` entries.

Errors: If not enough memory is available, an error may occur.

See also: `StringToPPChar` ([1047](#))

AsmFreemem

Synopsis: Routine that can be called from assembler routines to release memory.

Declaration: `procedure AsmFreemem(var p: pointer)`

Visibility: default

Description: `AsmFreemem` is a routine that can be called from assembler code to release previously allocated memory. The assembler reader cannot decide which overloaded `FreeMem` ([1003](#)) call should be used. `AsmFreeMem` provides a unique name that can be called from assembler. Other than that it is completely equivalent to `FreeMem`.

See also: `FreeMem` ([1003](#)), `AsmGetMem` ([976](#))

AsmGetmem

Synopsis: Routine that can be called from assembler routines to get memory.

Declaration: `procedure AsmGetmem(var p: pointer;size: PtrInt)`

Visibility: default

Description: `AsmGetmem` is a routine that can be called from assembler code to get memory. The assembler reader cannot decide which overloaded `GetMem` ([1005](#)) call should be used. `AsmGetMem` provides a unique name that can be called from assembler. Other than that it is completely equivalent to `GetMem`.

See also: `GetMem` ([1005](#)), `AsmFreeMem` ([976](#))

Assert

Synopsis: Check validity of a given condition.

Declaration: `procedure Assert(Expr: Boolean)`
`procedure Assert(Expr: Boolean; const Msg: String)`

Visibility: default

Description: With assertions on, `Assert` tests if `expr` is false, and if so, aborts the application with a Runtime error 227 and an optional error message in `msg`. If `expr` is true, program execution continues normally. If assertions are not enabled at compile time, this routine does nothing, and no code is generated for the `Assert` call. Enabling and disabling assertions at compile time is done via the `\$C` or `\$ASSERTIONS` compiler switches. These are global switches. The default behavior of the `assert` call can be changed by setting a new handler in the `AssertErrorProc` variable. `Sysutils` overrides the default handler to raise a `EAssertionFailed` exception.

Errors: None.

See also: `Halt` ([1007](#)), `Runerror` ([1036](#))

Assign

Synopsis: Assign a name to a file

Declaration: `procedure Assign(var f: File; const Name: String)`
`procedure Assign(var f: File; p: PChar)`
`procedure Assign(var f: File; c: Char)`
`procedure Assign(var f: TypedFile; const Name: String)`
`procedure Assign(var f: TypedFile; p: PChar)`
`procedure Assign(var f: TypedFile; c: Char)`
`procedure Assign(var t: Text; const s: String)`
`procedure Assign(var t: Text; p: PChar)`
`procedure Assign(var t: Text; c: Char)`

Visibility: default

Description: `Assign` assigns a name to `F`, which can be any file type. This call doesn't open the file, it just assigns a name to a file variable, and marks the file as closed.

Errors: None.

See also: `Reset` ([1033](#)), `Rewrite` ([1034](#)), `Append` ([975](#))

Listing: `./refex/ex5.pp`

Program `Example5`;

{ Program to demonstrate the Assign function. }

Var `F : text`;

begin

`Assign (F, '');`

Rewrite (`f`);

*{ The following can be put in any file by redirecting it
from the command line. }*

WriteLn (`f`, 'This goes to standard output !');

`Close (f);`

```

Assign (F, 'Test.txt');
rewrite (f);
writeln (f, 'This doesn't go to standard output !');
close (f);
end.

```

Assigned

Synopsis: Check if a pointer is valid

Declaration: `function Assigned(P: Pointer) : Boolean`

Visibility: default

Description: `Assigned` returns `True` if `P` is non-nil and returns `False` if `P` is nil. The main use of `Assigned` is that Procedural variables, method variables and class-type variables also can be passed to `Assigned`.

Errors: None

See also: `New` ([1022](#))

Listing: `./refex/ex96.pp`

Program `Example96`;

{ Program to demonstrate the Assigned function. }

Var `P : Pointer`;

```

begin
  If Not Assigned(P) then
    Writeln ('Pointer is initially NIL');
    P:=@P;
  If Not Assigned(P) then
    Writeln ('Internal inconsistency')
  else
    Writeln ('All is well in FPC')
end.

```

binStr

Synopsis: Convert integer to string with binary representation.

Declaration: `function binStr(Val: LongInt;cnt: Byte) : shortstring`
`function binStr(Val: Int64;cnt: Byte) : shortstring`

Visibility: default

Description: `BinStr` returns a string with the binary representation of `Value`. The string has at most `cnt` characters. (i.e. only the `cnt` rightmost bits are taken into account) To have a complete representation of any longint-type value, 32 bits are needed, i.e. `cnt=32`

Errors: None.

See also: `Str` ([1046](#)), `Val` ([1054](#)), `HexStr` ([1008](#)), `OctStr` ([1023](#))

Listing: ./refex/ex82.pp

```

Program example82;

{ Program to demonstrate the BinStr function }

Const Value = 45678;

Var I : longint;

begin
  For I:=8 to 20 do
    WriteLn ( BinStr(Value,I):20);
end.
```

BlockRead

Synopsis: Read data from an untyped file into memory

Declaration:

```

procedure BlockRead(var f: File; var Buf; count: LongInt;
                    var Result: LongInt)
procedure BlockRead(var f: File; var Buf; count: Cardinal;
                    var Result: Cardinal)
procedure BlockRead(var f: File; var Buf; count: Word; var Result: Word)
procedure BlockRead(var f: File; var Buf; count: Word; var Result: Integer)
procedure BlockRead(var f: File; var Buf; count: LongInt)
```

Visibility: default

Description: Blockread reads count or less records from file F. A record is a block of bytes with size specified by the Rewrite (1034) or Reset (1033) statement. The result is placed in Buffer, which must contain enough room for Count records. The function cannot read partial records. If Result is specified, it contains the number of records actually read. If Result isn't specified, and less than Count records were read, a run-time error is generated. This behavior can be controlled by the \var{\{\$i\}} switch.

Errors: Depending on the state of the \var{\{\$I\}} switch, a runtime error can be generated if there is an error. In the \var{\{\$I-\}} state, use IOResult to check for errors.

See also: Blockwrite (980), Close (982), Reset (1033), Assign (977)

Listing: ./refex/ex6.pp

```

Program Example6;

{ Program to demonstrate the BlockRead and BlockWrite functions. }

Var Fin, fout : File;
    NumRead, NumWritten : Word;
    Buf : Array[1..2048] of byte;
    Total : Longint;

begin
  Assign (Fin, Paramstr(1));
  Assign (Fout, Paramstr(2));
  Reset (Fin, 1);
  Rewrite (Fout, 1);
```

```

Total:=0;
Repeat
  BlockRead ( Fin , buf , Sizeof( buf ) , NumRead);
  BlockWrite ( Fout , Buf , NumRead , NumWritten);
  inc( Total , NumWritten);
Until (NumRead=0) or (NumWritten<>NumRead);
Write ( 'Copied ' , Total , ' bytes from file ' , paramstr(1));
WriteLn ( ' to file ' , paramstr(2));
close( fin );
close( fout );
end.

```

BlockWrite

Synopsis: Write data from memory to an untyped file

Declaration: `procedure BlockWrite(var f: File; const Buf; Count: LongInt; var Result: LongInt)`
`procedure BlockWrite(var f: File; const Buf; Count: Cardinal; var Result: Cardinal)`
`procedure BlockWrite(var f: File; const Buf; Count: Word; var Result: Word)`
`procedure BlockWrite(var f: File; const Buf; Count: Word; var Result: Integer)`
`procedure BlockWrite(var f: File; const Buf; Count: LongInt)`

Visibility: default

Description: `BlockWrite` writes count records from buffer to the file F. A record is a block of bytes with size specified by the `Rewrite` (1034) or `Reset` (1033) statement. If the records couldn't be written to disk, a run-time error is generated. This behavior can be controlled by the `\var{\{$i\}}` switch.

Errors: Depending on the state of the `\var{\{$I\}}` switch, a runtime error can be generated if there is an error. In the `\var{\{$I-\}}` state, use `IOResult` to check for errors.

See also: `Blockread` (979), `Close` (982), `Rewrite` (1034), `Assign` (977)

Break

Synopsis: Exit current loop construct.

Declaration: `procedure Break`

Visibility: default

Description: `Break` jumps to the statement following the end of the current repetitive statement. The code between the `Break` call and the end of the repetitive statement is skipped. The condition of the repetitive statement is NOT evaluated.

This can be used with `For`, `var{repeat}` and `While` statements.

Note that while this is a procedure, `Break` is a reserved word and hence cannot be redefined.

Errors: None.

See also: `Continue` (988), `Exit` (997)

Listing: ./refex/ex87.pp

```

Program Example87;

{ Program to demonstrate the Break function . }

Var I : longint;

begin
  I:=0;
  While I <10 Do
    begin
      Inc(I);
      If I >5 Then
        Break;
      Writeln ( i );
    end;
  I:=0;
  Repeat
    Inc(I);
    If I >5 Then
      Break;
    Writeln ( i );
  Until I >=10;
  For I:=1 to 10 do
    begin
      If I >5 Then
        Break;
      Writeln ( i );
    end;
end.

```

chdir

Synopsis: Change current working directory.

Declaration: `procedure chdir(const s: String)`

Visibility: default

Description: Chdir changes the working directory of the process to S.

Errors: Depending on the state of the `\var{\{$I\}}` switch, a runtime error can be generated if there is an error. In the `\var{\{$I-\}}` state, use `IOResult` to check for errors.

See also: Mkdir ([1021](#)), Rmdir ([1035](#))

Listing: `./refex/ex7.pp`

```

Program Example7;

{ Program to demonstrate the ChDir function . }

begin
  {$I-}
  ChDir ( ParamStr(1));
  if IOResult <>0 then
    Writeln ( 'Cannot change to directory : ', paramstr (1));
end.

```

Chr

Synopsis: Convert byte value to character value

```
Declaration: function Chr(b: Byte) : Char
```

Visibility: default

Description: Chr returns the character which has ASCII value X.

Errors: None.

See also: Ord (1024), Str (1046)

Listing: ./refex/ex8.pp

Program Example8;

```
{ Program to demonstrate the Chr function. }
```

begin

```
Write (chr(10),chr(13)); { The same effect as WriteLn; }  
end.
```

Close

Synopsis: Close a file

```
Declaration: procedure Close(var f: File)
             procedure Close(var t: Text)
```

Visibility: default

Description: Close flushes the buffer of the file F and closes F. After a call to Close, data can no longer be read from or written to F. To reopen a file closed with Close, it isn't necessary to assign the file again. A call to Reset (1033) or Rewrite (1034) is sufficient.

Errors: Depending on the state of the `\var{\{\$I\}}` switch, a runtime error can be generated if there is an error. In the `\var{\{\$I-\}}` state, use `IOResult` to check for errors.

See also: [Assign \(977\)](#), [Reset \(1033\)](#), [Rewrite \(1034\)](#), [Flush \(1002\)](#)

Listing: ./refex/ex9.pp

Program Example9;

```
{ Program to demonstrate the Close function. }
```

```
Var F : text;
```

begin

```
Assign (f, 'Test.txt');
ReWrite (F);
WriteIn (F, 'Some text written to Test.txt');
close (f); { Flushes contents of buffer to disk,
             closes the file. Omitting this may
             cause data NOT to be written to disk.}
```

end.

CompareByte

Synopsis: Compare 2 memory buffers byte per byte

Declaration: `function CompareByte(const buf1;const buf2;len: SizeInt) : SizeInt`

Visibility: default

Description: `CompareByte` compares two memory regions `buf1`, `buf2` on a byte-per-byte basis for a total of `len` bytes.

The function returns one of the following values:

less than 0 if `buf1` and `buf2` contain different bytes in the first `len` bytes, and the first such byte is smaller in `buf1` than the byte at the same position in `buf2`.

0 if the first `len` bytes in `buf1` and `buf2` are equal. **> 0** if `buf1` and `buf2` contain different bytes in the first `len` bytes, and the first such byte is larger in `buf1` than the byte at the same position in `buf2`.

Errors: None.

See also: `CompareChar` (984), `CompareWord` (986), `CompareDWord` (985)

Listing: `./refex/ex99.pp`

Program `Example99`;

{ Program to demonstrate the CompareByte function. }

Const

`ArraySize = 100;`
`HalfArraySize = ArraySize Div 2;`

Var

`Buf1, Buf2 : Array[1..ArraySize] of byte;`
`I : longint;`

Procedure `CheckPos(Len : Longint);`

Begin

`Write('First ', Len, ' positions are ');`
`if CompareByte(Buf1, Buf2, Len) <> 0 then`
`Write('NOT ');`
`Writeln('equal');`
`end;`

begin

`For I:=1 to ArraySize do`
`begin`
`Buf1[I]:= I;`
`If I<=HalfArraySize Then`
`Buf2[I]:= I`
`else`
`Buf2[I]:= HalfArraySize-I;`
`end;`
`CheckPos(HalfArraySize div 2);`
`CheckPos(HalfArraySize);`
`CheckPos(HalfArraySize+1);`
`CheckPos(HalfArraySize + HalfArraySize Div 2);`
`end.`

CompareChar

Synopsis: ompare 2 memory buffers character per character

Declaration: `function CompareChar(const buf1;const buf2;len: SizeInt) : SizeInt`

Visibility: default

Description: CompareChar compares two memory regions buf1,buf2 on a character-per-character basis for a total of len characters.

The CompareChar0 variant compares len bytes, or until a zero character is found.

The function returns one of the following values:

-1if buf1 and buf2 contain different characters in the first len positions, and the first such character is smaller in buf1 than the character at the same position in buf2.

0if the first len characters in buf1 and buf2 are equal.

1if buf1 and buf2 contain different characters in the first len positions, and the first such character is larger in buf1 than the character at the same position in buf2.

Errors: None.

See also: CompareByte (983), CompareWord (986), CompareDWord (985)

Listing: ./refex/ex100.pp

Program Example100;

{ Program to demonstrate the CompareChar function. }

Const

 ArraySize = 100;
 HalfArraySize = ArraySize **Div** 2;

Var

 Buf1,Buf2 : **Array**[1..ArraySize] **of** char;
 I : longint;

Procedure CheckPos(Len : Longint);

Begin

Write('First ',Len,' characters are ');
 if CompareChar(Buf1,Buf2,Len)<>0 **then**
 Write('NOT ');
 Writeln('equal');
end;

Procedure CheckNullPos(Len : Longint);

Begin

Write('First ',Len,' non-null characters are ');
 if CompareChar0(Buf1,Buf2,Len)<>0 **then**
 Write('NOT ');
 Writeln('equal');
end;

begin

For I:=1 **to** ArraySize **do**
 begin

```

    Buf1[i]:=chr(l);
    If l<=HalfArraySize Then
        Buf2[l]:=chr(l)
    else
        Buf2[i]:=chr(HalfArraySize-l);
    end;
    CheckPos(HalfArraySize div 2);
    CheckPos(HalfArraySize);
    CheckPos(HalfArraySize+1);
    CheckPos(HalfArraySize + HalfArraySize Div 2);
    For l:=1 to 4 do
        begin
            buf1[Random(ArraySize)+1]:=Chr(0);
            buf2[Random(ArraySize)+1]:=Chr(0);
        end;
        Randomize;
        CheckNullPos(HalfArraySize div 2);
        CheckNullPos(HalfArraySize);
        CheckNullPos(HalfArraySize+1);
        CheckNullPos(HalfArraySize + HalfArraySize Div 2);
    end.

```

CompareChar0

Synopsis: Compare two buffers character by character till a null-character is reached.

Declaration: `function CompareChar0(const buf1;const buf2;len: SizeInt) : SizeInt`

Visibility: default

Description: CompareChar0 compares 2 buffers buf1 and buf2 for a maximum length of len or till a null character is reached in either buffer. The result depends on the contents of the buffers:

- < 0 If buf1 contains a character less than the corresponding character in buf2.
- 0 If both buffers are equal
- > 0 If buf1 contains a character greater than the corresponding character in buf2.

Errors: None.

See also: CompareByte ([983](#)), CompareChar ([984](#)), CompareDWord ([985](#)), CompareWord ([986](#))

CompareDWord

Synopsis: Compare 2 memory buffers DWord per DWord

Declaration: `function CompareDWord(const buf1;const buf2;len: SizeInt) : SizeInt`

Visibility: default

Description: CompareDWord compares two memory regions buf1,buf2 on a DWord-per-DWord basis for a total of len DWords. (A DWord is 4 bytes).

The function returns one of the following values:

- 1 if buf1 and buf2 contain different DWords in the first len DWords, and the first such DWord is smaller in buf1 than the DWord at the same position in buf2.

0If the first len DWords in buf1 and buf2 are equal.

1If buf1 and buf2 contain different DWords in the first len DWords, and the first such DWord is larger in buf1 than the DWord at the same position in buf2.

Errors: None.

See also: CompareChar (984), CompareByte (983), CompareWord (986)

Listing: ./refex/ex101.pp

Program Example101;

{ Program to demonstrate the CompareDWord function. }

Const

 ArraySize = 100;
 HalfArraySize = ArraySize **Div** 2;

Var

 Buf1, Buf2 : **Array**[1..ArraySize] **of** Dword;
 I : longint;

Procedure CheckPos(Len : Longint);

Begin

Write('First ', Len, ' DWords are ');
 if CompareDWord(Buf1, Buf2, Len) <> 0 **then**
 Write('NOT ');
 Writeln('equal');
end;

begin

For I:=1 **to** ArraySize **do**
 begin
 Buf1[I]:= I;
 If I<=HalfArraySize **Then**
 Buf2[I]:= I
 else
 Buf2[I]:= HalfArraySize-I;
 end;
 CheckPos(HalfArraySize **div** 2);
 CheckPos(HalfArraySize);
 CheckPos(HalfArraySize+1);
 CheckPos(HalfArraySize + HalfArraySize **Div** 2);
 end.

CompareWord

Synopsis: Compare 2 memory buffers word per word

Declaration: function CompareWord(const buf1;const buf2;len: SizeInt) : SizeInt

Visibility: default

Description: CompareWord compares two memory regions buf1,buf2 on a Word-per-Word basis for a total of len Words. (A Word is 2 bytes).

The function returns one of the following values:

-If buf1 and buf2 contain different Words in the first len Words, and the first such Word is smaller in buf1 than the Word at the same position in buf2.

0If the first len Words in buf1 and buf2 are equal.

If buf1 and buf2 contain different Words in the first len Words, and the first such Word is larger in buf1 than the Word at the same position in buf2.

Errors: None.

See also: CompareChar (984), CompareByte (983), CompareDWord (985)

Listing: ./refex/ex102.pp

Program Example102;

{ Program to demonstrate the CompareWord function. }

Const

ArraySize = 100;
HalfArraySize = ArraySize **Div** 2;

Var

Buf1, Buf2 : **Array**[1..ArraySize] **of** Word;
I : longint;

Procedure CheckPos(Len : Longint);

Begin

Write('First ', Len, ' words are ');
if CompareWord(Buf1, Buf2, Len) <> 0 **then**
 Write('NOT ');
 Writeln('equal ');
end;

begin

For I:=1 **to** ArraySize **do**
 begin
 Buf1[I]:=I;
 If I<=HalfArraySize **Then**
 Buf2[I]:=I
 else
 Buf2[I]:=HalfArraySize-I;
 end;
 CheckPos(HalfArraySize **div** 2);
 CheckPos(HalfArraySize);
 CheckPos(HalfArraySize+1);
 CheckPos(HalfArraySize + HalfArraySize **Div** 2);
end.

Concat

Synopsis: Append one string to another.

Declaration: function Concat(const S1: String;const S2: String;const S3: String;
 const Sn: String) : String

Visibility: default

Description: Concat concatenates the strings S1,S2 etc. to one long string. The resulting string is truncated at a length of 255 bytes. The same operation can be performed with the + operation.

Errors: None.

See also: Copy ([989](#)), Delete ([991](#)), Insert ([1015](#)), Pos ([1026](#)), Length ([1017](#))

Listing: ./refex/ex10.pp

Program Example10;

{ Program to demonstrate the Concat function . }

Var

S : String;

begin

S:=Concat('This can be done',' Easier ','with the + operator !');

end.

Continue

Synopsis: Continue with next loop cycle.

Declaration: procedure Continue

Visibility: default

Description: Continue jumps to the end of the current repetitive statement. The code between the Continue call and the end of the repetitive statement is skipped. The condition of the repetitive statement is then checked again.

This can be used with For, var{repeat} and While statements.

Note that while this is a procedure, Continue is a reserved word and hence cannot be redefined.

Errors: None.

See also: Break ([980](#)), Exit ([997](#))

Listing: ./refex/ex86.pp

Program Example86;

{ Program to demonstrate the Continue function . }

Var I : longint;

begin

I:=0;

While I < 10 **Do**

begin

Inc(I);

If I < 5 **Then**

Continue;

WriteLn (i);

end;

I:=0;

Repeat

Inc(I);

```

    If I < 5 Then
        Continue;
    Writeln (i);
    Until I >= 10;
    For I := 1 to 10 do
        begin
            If I < 5 Then
                Continue;
            Writeln (i);
        end;
    end.

```

copy

Synopsis: Copy part of a string.

Declaration: `function Copy(const s: shortstring; index: SizeInt; count: SizeInt) : shortstring`
`function copy(c: Char; index: SizeInt; count: SizeInt) : shortstring`
`function Copy(const S: AnsiString; Index: SizeInt; Size: SizeInt) : AnsiString`

Visibility: default

Description: Copy returns a string which is a copy of the Count characters in S, starting at position Index. If Count is larger than the length of the string S, the result is truncated. If Index is larger than the length of the string S, then an empty string is returned.

Errors: None.

See also: Delete ([991](#)), Insert ([1015](#)), Pos ([1026](#))

Listing: ./refex/ex11.pp

```

Program Example11;

{ Program to demonstrate the Copy function. }

Var S,T : String;

begin
    T:= '1234567';
    S:=Copy (T,1,2);    { S:= '12' }
    S:=Copy (T,4,2);    { S:= '45' }
    S:=Copy (T,4,8);    { S:= '4567' }
end.

```

COS

Synopsis: Calculate cosine of angle

Declaration: `function cos(d: ValReal) : ValReal`

Visibility: default

Description: Cos returns the cosine of X, where X is an angle, in radians. If the absolute value of the argument is larger than $\sqrt{2} \times 10^3$, then the result is undefined.

Errors: None.

See also: Arctan ([975](#)), Sin ([1043](#))

Listing: ./refex/ex12.pp

```
Program Example12;

{ Program to demonstrate the Cos function. }

Var R : Real;

begin
  R:=Cos(Pi);      { R:=-1 }
  R:=Cos(Pi/2);    { R:=0  }
  R:=Cos(0);       { R:=1  }
end.
```

Cseg

Synopsis: Return code segment

Declaration: function Cseg : Word

Visibility: default

Description: CSeg returns the Code segment register. In Free Pascal, it returns always a zero, since Free Pascal is a 32 bit compiler.

Errors: None.

See also: DSeg ([993](#)), Seg ([1038](#)), OfS ([1024](#)), Ptr ([1028](#))

Listing: ./refex/ex13.pp

```
Program Example13;

{ Program to demonstrate the CSeg function. }

var W : word;

begin
  W:=CSeg; {W:=0, provided for compatibility ,
           FPC is 32 bit.}
end.
```

Dec

Synopsis: Decrease value of variable

Declaration: procedure Dec(var X: TOrdinal)
 procedure Dec(var X: TOrdinal;Decrement: TOrdinal)

Visibility: default

Description: Dec decreases the value of X with Decrement. If Decrement isn't specified, then 1 is taken as a default.

Errors: A range check can occur, or an underflow error, if an attempt it made to decrease X below its minimum value.

See also: Inc ([1010](#))

Listing: ./refex/ex14.pp

Program Example14;

{ Program to demonstrate the Dec function. }

Var

I : Integer;
L : Longint;
W : Word;
B : Byte;
Si : ShortInt;

begin

I:=1;
L:=2;
W:=3;
B:=4;
Si:=5;
Dec (i); { i:=0 }
Dec (L,2); { L:=0 }
Dec (W,2); { W:=1 }
Dec (B, -2); { B:=6 }
Dec (Si ,0); { Si:=5 }
end.

Delete

Synopsis: Delete part of a string.

Declaration: `procedure Delete(var s: shortstring; index: SizeInt; count: SizeInt)`
`procedure Delete(var S: AnsiString; Index: SizeInt; Size: SizeInt)`

Visibility: default

Description: Delete removes Count characters from string S, starting at position Index. All characters after the deleted characters are shifted Count positions to the left, and the length of the string is adjusted.

Errors: None.

See also: Copy ([989](#)), Pos ([1026](#)), Insert ([1015](#))

Listing: ./refex/ex15.pp

Program Example15;

{ Program to demonstrate the Delete function. }

Var

S : **String**;

begin

S:= 'This is not easy !';

```
Delete (S,9,4); { S:='This is easy !' }
end.
```

Dispose

Synopsis: Free dynamically allocated memory

Declaration: `procedure Dispose(P: Pointer)`
`procedure Dispose(P: TypedPointer;Des: TProcedure)`

Visibility: default

Description: The first form `Dispose` releases the memory allocated with a call to `New` (1022). The pointer `P` must be typed. The released memory is returned to the heap.

The second form of `Dispose` accepts as a first parameter a pointer to an object type, and as a second parameter the name of a destructor of this object. The destructor will be called, and the memory allocated for the object will be freed.

Errors: An runtime error will occur if the pointer doesn't point to a location in the heap.

See also: `New` (1022), `Getmem` (1005), `Freemem` (1003)

Listing: ./refex/ex16.pp

Program Example16;

{ Program to demonstrate the Dispose and New functions. }

Type SS = **String**[20];

```
  AnObj = Object
    I : integer;
    Constructor Init;
    Destructor Done;
  end;
```

Var

```
  P : ^SS;
  T : ^AnObj;
```

Constructor Anobj.Init;

begin

```
  WriteLn ('Initializing an instance of AnObj !');
end;
```

Destructor AnObj.Done;

begin

```
  WriteLn ('Destroying an instance of AnObj !');
end;
```

begin

```
  New (P);
  P^:= 'Hello , World !';
  Dispose (P);
  { P is undefined from here on !}
```

```

New(T, Init);
T^.i:=0;
Dispose (T, Done);
end.

```

Dseg

Synopsis: Return data segment

Declaration: `function Dseg : Word`

Visibility: default

Description: DSeg returns the data segment register. In Free Pascal, it returns always a zero, since Free Pascal is a 32 bit compiler.

Errors: None.

See also: CSeg ([990](#)), Seg ([1038](#)), OfS ([1024](#)), Ptr ([1028](#))

Listing: ./refex/ex17.pp

Program Example17;

{ Program to demonstrate the DSeg function. }

Var

W : Word;

begin

W:=**DSeg**; *{W:=0, This function is provided for compatibility,
FPC is a 32 bit comiler.}*

end.

Dump_Stack

Synopsis: Dump stack to the given text file.

Declaration: `procedure Dump_Stack(var f: text;bp: pointer)`

Visibility: default

Description: Dump_Stack prints a stack dump to the file f, with base frame pointer bp

Errors: The file f must be opened for writing or an error will occur.

See also: get_caller_addr ([1006](#)), get_caller_frame ([1007](#)), get_frame ([1007](#))

EOF

Synopsis: Check for end of file

Declaration: `function EOF(var f: File) : Boolean`
`function EOF(var t: Text) : Boolean`
`function EOF : Boolean`

Visibility: default

Description: `Eof` returns `True` if the file-pointer has reached the end of the file, or if the file is empty. In all other cases `Eof` returns `False`. If no file `F` is specified, standard input is assumed.

Errors: Depending on the state of the `\var{\{\$I\}}` switch, a runtime error can be generated if there is an error. In the `\var{\{\$I-\}}` state, use `IOResult` to check for errors.

See also: `Eoln` ([994](#)), `Assign` ([977](#)), `Reset` ([1033](#)), `Rewrite` ([1034](#))

Listing: `./refex/ex18.pp`

Program `Example18`;

{ Program to demonstrate the Eof function. }

Var `T1,T2 : text;`
 `C : Char;`

begin
 { Set file to read from. Empty means from standard input. }
 `assign (t1,paramstr(1));`
 `reset (t1);`
 { Set file to write to. Empty means to standard output. }
 `assign (t2,paramstr(2));`
 `rewrite (t2);`
 While not eof(t1) **do**
 begin
 `read (t1,C);`
 `write (t2,C);`
 end;
 `Close (t1);`
 `Close (t2);`
end.

EOLn

Synopsis: Check for end of line

Declaration: `function EOLn(var t: Text) : Boolean`
 `function EOLn : Boolean`

Visibility: default

Description: `Eof` returns `True` if the file pointer has reached the end of a line, which is demarcated by a line-feed character (ASCII value 10), or if the end of the file is reached. In all other cases `Eof` returns `False`. If no file `F` is specified, standard input is assumed. It can only be used on files of type `Text`.

Errors: None.

See also: `Eof` ([993](#)), `Assign` ([977](#)), `Reset` ([1033](#)), `Rewrite` ([1034](#))

Listing: `./refex/ex19.pp`

Program `Example19`;

{ Program to demonstrate the Eoln function. }

```

begin
  { This program waits for keyboard input. }
  { It will print True when an empty line is put in ,
    and false when you type a non-empty line.
    It will only stop when you press enter. }
  While not Eoln do
    Writeln (eoln);
end.

```

Erase

Synopsis: Delete a file from disk

Declaration: `procedure Erase(var f: File)`
`procedure Erase(var t: Text)`

Visibility: default

Description: Erase removes an unopened file from disk. The file should be assigned with Assign, but not opened with Reset or Rewrite

Errors: Depending on the state of the `\var{\{$I\}}` switch, a runtime error can be generated if there is an error. In the `\var{\{$I-\}}` state, use `IOResult` to check for errors.

See also: Assign ([977](#))

Listing: `./refex/ex20.pp`

Program Example20;

```
{ Program to demonstrate the Erase function. }
```

```
Var F : Text;
```

```

begin
  { Create a file with a line of text in it }
  Assign (F, 'test.txt');
  Rewrite (F);
  Writeln (F, 'Try and find this when I 'm finished !');
  close (f);
  { Now remove the file }
  Erase (f);
end.

```

Exclude

Synopsis: Exclude element from a set if it is present.

Declaration: `procedure Exclude(var S: TSetType; E: TSetElement)`

Visibility: default

Description: Exclude removes E from the set S if it is included in the set. E should be of the same type as the base type of the set S.

Thus, the two following statements do the same thing:


```
S:=S-[E];
Exclude(S,E);
```

Errors: If the type of the element E is not equal to the base type of the set S , the compiler will generate an error.

See also: Include ([1011](#))

Listing: ./refex/ex111.pp

```
program Example111;

{ Program to demonstrate the Include/Exclude functions }

Type
TEnumA = (aOne,aTwo,aThree);
TEnumAs = Set of TEnumA;

Var
SA : TEnumAs;

Procedure PrintSet(S : TEnumAs);

var
B : Boolean;

procedure DoEI(A : TEnumA; Desc : String);

begin
  If A in S then
    begin
      If B then
        Write( ' , ');
        B:=True;
        Write(Desc);
      end;
    end;

begin
  Write( ' [ ');
  B:=False;
  DoEI(aOne, 'aOne');
  DoEI(aTwo, 'aTwo');
  DoEI(aThree, 'aThree');
  WriteLn( ' ] ');
end;

begin
  SA:=[];
  Include(SA,aOne);
  PrintSet(SA);
  Include(SA,aThree);
  PrintSet(SA);
  Exclude(SA,aOne);
  PrintSet(SA);
  Exclude(SA,aTwo);
  PrintSet(SA);
  Exclude(SA,aThree);
```

```

    PrintSet(SA);
end.

```

Exit

Synopsis: Exit current subroutine.

Declaration: `procedure Exit(const X: TAnyType)`
`procedure Exit`

Visibility: default

Description: `Exit` exits the current subroutine, and returns control to the calling routine. If invoked in the main program routine, `exit` stops the program. The optional argument `X` allows to specify a return value, in the case `Exit` is invoked in a function. The function result will then be equal to `X`.

Errors: None.

See also: `Halt` ([1007](#))

Listing: `./refex/ex21.pp`

Program `Example21`;

{ Program to demonstrate the Exit function. }

Procedure `DoAnExit (Yes : Boolean);`

{ This procedure demonstrates the normal Exit }

begin

Writeln ('Hello from DoAnExit !');

If Yes **then**

begin

Writeln ('Bailing out early.');

exit;

end;

Writeln ('Continuing to the end.');

end;

Function `Positive (Which : Integer) : Boolean;`

*{ This function demonstrates the extra FPC feature of Exit :
 You can specify a return value for the function }*

begin

if Which > 0 **then**

exit (True)

else

exit (False);

end;

begin

{ This call will go to the end }

 DoAnExit (False);

{ This call will bail out early }

 DoAnExit (True);

if Positive (-1) **then**

```

    Writeln ( 'The compiler is nuts, -1 is not positive.')
  else
    Writeln ( 'The compiler is not so bad, -1 seems to be negative. ');
end.

```

exp

Synopsis: Exponentiate

Declaration: `function exp(d: ValReal) : ValReal`

Visibility: default

Description: Exp returns the exponent of X, i.e. the number e to the power X.

Errors: None.

See also: Ln ([1018](#)), Power ([1027](#))

Listing: ./refex/ex22.pp

Program Example22;

{ Program to demonstrate the Exp function. }

begin

```

  Writeln (Exp(1):8:2); { Should print 2.72 }
end.

```

FilePos

Synopsis: Get position in file

Declaration: `function FilePos(var f: File) : LongInt`

Visibility: default

Description: Filepos returns the current record position of the file-pointer in file F. It cannot be invoked with a file of type Text. A compiler error will be generated if this is attempted.

Errors: Depending on the state of the `\var{\{$I\}}` switch, a runtime error can be generated if there is an error. In the `\var{\{$I-\}}` state, use `IOResult` to check for errors.

See also: Filesize ([999](#))

Listing: ./refex/ex23.pp

Program Example23;

{ Program to demonstrate the FilePos function. }

```

Var F : File of Longint;
    L,FP : longint;

```

begin

```

  { Fill a file with data :
    Each position contains the position ! }

```

```

Assign (F, 'test.tmp');
Rewrite (F);
For L:=0 to 100 do
  begin
    FP:=FilePos(F);
    Write (F,FP);
  end;
Close (F);
Reset (F);
{ If all goes well, nothing is displayed here. }
While not (Eof(F)) do
  begin
    FP:=FilePos (F);
    Read (F,L);
    if L<>FP then
      Writeln ( 'Something wrong: Got ',L, ' on pos ',FP);
    end;
  end;
Close (F);
Erase (f);
end.

```

FileSize

Synopsis: Size of file

Declaration: function FileSize(var f: File) : LongInt

Visibility: default

Description: Filesize returns the total number of records in file F. It cannot be invoked with a file of type Text. (under linux and unix, this also means that it cannot be invoked on pipes). If F is empty, 0 is returned.

Errors: Depending on the state of the `\var{\{$I\}}` switch, a runtime error can be generated if there is an error. In the `\var{\{$I-\}}` state, use `IOResult` to check for errors.

See also: Filepos ([998](#))

Listing: ./refex/ex24.pp

Program Example24;

{ Program to demonstrate the FileSize function. }

```

Var F : File Of byte;
    L : File Of Longint;

begin
  Assign (F,paramstr(1));
  Reset (F);
  Writeln ( 'File size in bytes : ',FileSize(F));
  Close (F);
  Assign (L,paramstr (1));
  Reset (L);
  Writeln ( 'File size in Longints : ',FileSize(L));
  Close (f);
end.

```

FillByte

Synopsis: Fill memory region with 8-bit pattern

Declaration: `procedure FillByte(var x; count: SizeInt; value: Byte)`

Visibility: default

Description: `FillByte` fills the memory starting at `X` with `Count` bytes with value equal to `Value`. This is useful for quickly zeroing out a memory location. When the size of the memory location to be filled out is a multiple of 2 bytes, it is better to use `Fillword` (1002), and if it is a multiple of 4 bytes it is better to use `FillDWord` (1001), these routines are optimized for their respective sizes.

Errors: No checking on the size of `X` is done.

See also: `Fillchar` (1000), `FillDWord` (1001), `Fillword` (1002), `Move` (1021)

Listing: `./refex/ex102.pp`

Program `Example102;`

{ Program to demonstrate the CompareWord function. }

Const

`ArraySize = 100;
 HalfArraySize = ArraySize Div 2;`

Var

`Buf1, Buf2 : Array[1..ArraySize] of Word;
 I : longint;`

Procedure `CheckPos(Len : Longint);`

Begin

`Write('First ', Len, ' words are ');
 if CompareWord(Buf1, Buf2, Len) <> 0 then
 Write('NOT ');
 Writeln('equal');
 end;`

begin

`For I:=1 to ArraySize do
 begin
 Buf1[I]:=I;
 If I<=HalfArraySize Then
 Buf2[I]:=I
 else
 Buf2[I]:=HalfArraySize-I;
 end;
 CheckPos(HalfArraySize div 2);
 CheckPos(HalfArraySize);
 CheckPos(HalfArraySize+1);
 CheckPos(HalfArraySize + HalfArraySize Div 2);
 end.`

FillChar

Synopsis: Fill memory region with certain character

Declaration: `procedure FillChar(var x;count: SizeInt;Value: Boolean)`
`procedure FillChar(var x;count: SizeInt;Value: Char)`
`procedure FillChar(var x;count: SizeInt;Value: Byte)`

Visibility: default

Description: Fillchar fills the memory starting at X with Count bytes or characters with value equal to Value.

Errors: No checking on the size of X is done.

See also: Fillword ([1002](#)), Move ([1021](#)), FillByte ([1000](#)), FillDWord ([1001](#))

Listing: ./refex/ex25.pp

Program Example25;

{ Program to demonstrate the FillChar function. }

```
Var S : String[10];
    I : Byte;
begin
  For i:=10 downto 0 do
    begin
      { Fill S with i spaces }
      FillChar (S,SizeOf(S),' ');
      { Set Length }
      SetLength(S,I);
      Writeln (s,'*');
    end;
end.
```

FillDWord

Synopsis: Fill memory region with 32-bit pattern

Declaration: `procedure FillDWord(var x;count: SizeInt;value: DWord)`

Visibility: default

Description: Fillword fills the memory starting at X with Count DWords with value equal to Value. A DWord is 4 bytes in size.

Errors: No checking on the size of X is done.

See also: FillByte ([1000](#)), Fillchar ([1000](#)), Fillword ([1002](#)), Move ([1021](#))

Listing: ./refex/ex103.pp

Program Example103;

{ Program to demonstrate the FillByte function. }

```
Var S : String[10];
    I : Byte;

begin
  For i:=10 downto 0 do
```

```

begin
  { Fill S with i bytes }
  FillChar (S, SizeOf(S), 32);
  { Set Length }
  SetLength(S, I);
  Writeln (s, '*');
end;
end.

```

FillWord

Synopsis: Fill memory region with 16-bit pattern

Declaration: `procedure FillWord(var x; count: SizeInt; Value: Word)`

Visibility: default

Description: Fillword fills the memory starting at X with Count words with value equal to Value. A word is 2 bytes in size.

Errors: No checking on the size of X is done.

See also: Fillchar ([1000](#)), Move ([1021](#))

Listing: ./refex/ex76.pp

Program Example76;

```
{ Program to demonstrate the FillWord function. }
```

```
Var W : Array[1..100] of Word;
```

```
begin
```

```
  { Quick initialization of array W }
```

```
  FillWord(W, 100, 0);
```

```
end.
```

Flush

Synopsis: Write file buffers to disk

Declaration: `procedure Flush(var t: Text)`

Visibility: default

Description: Flush empties the internal buffer of an opened file F and writes the contents to disk. The file is \textit{not} closed as a result of this call.

Errors: Depending on the state of the \var{\{\\$I\}} switch, a runtime error can be generated if there is an error. In the \var{\{\\$I-\}} state, use IOResult to check for errors.

See also: Close ([982](#))

Listing: ./refex/ex26.pp

Program Example26;

{ Program to demonstrate the Flush function. }

Var F : Text;

begin

{ Assign F to standard output }

Assign (F, '');

Rewrite (F);

Writeln (F, 'This line is written first , but appears later !');

*{ At this point the text is in the internal pascal buffer,
and not yet written to standard output }*

Writeln ('This line appears first , but is written later !');

*{ A writeln to 'output' always causes a flush – so this text is
written to screen }*

Flush (f);

{ At this point , the text written to F is written to screen. }

Write (F, 'Finishing ');

Close (f); *{ Closing a file always causes a flush first }*

Writeln ('off.');

end.

frac

Synopsis: Return fractional part of floating point value.

Declaration: function frac(d: ValReal) : ValReal

Visibility: default

Description: Frac returns the non-integer part of X.

Errors: None.

See also: Round ([1035](#)), Int ([1015](#))

Listing: ./refex/ex27.pp

Program Example27;

{ Program to demonstrate the Frac function. }

Var R : Real;

begin

Writeln (Frac (123.456):0:3); *{ Prints 0.456 }*

Writeln (Frac (-123.456):0:3); *{ Prints -0.456 }*

end.

Freemem

Synopsis: Release allocated memory

Declaration: procedure Freemem(p: pointer;Size: PtrInt)
function Freemem(p: pointer) : PtrInt

Visibility: default

Description: `FreeMem` releases the memory occupied by the pointer `P`, of size `Count` (in bytes), and returns it to the heap. `P` should point to the memory allocated to a dynamic variable.

Errors: An error will occur when `P` doesn't point to the heap.

See also: `GetMem` ([1005](#)), `New` ([1022](#)), `Dispose` ([992](#))

Listing: `./refex/ex28.pp`

Program `Example28`;

{ Program to demonstrate the FreeMem and GetMem functions. }

Var `P` : `Pointer`;
 `MM` : `Longint`;

begin

{ Get memory for P }

`MM:=MemAvail`;

`WriteLn` ('Memory available before GetMem : ',**MemAvail**);

`GetMem` (`P`,80);

`MM:=MM-Memavail`;

`Write` ('Memory available after GetMem : ',**MemAvail**);

`WriteLn` (' or ',MM,' bytes less than before the call.');

{ fill it with spaces }

`FillChar` (`P`^,80,' ');

{ Free the memory again }

`FreeMem` (`P`,80);

`WriteLn` ('Memory available after FreeMem : ',**MemAvail**);

end.

Freememory

Synopsis: Alias for `FreeMem` ([1003](#))

Declaration: `procedure Freememory(p: pointer;Size: PtrInt)`
 `function Freememory(p: pointer) : PtrInt`

Visibility: default

Description: `FreeMemory` is an alias for `FreeMem` ([1003](#)).

See also: `FreeMem` ([1003](#))

getdir

Synopsis: Return the current directory

Declaration: `procedure getdir(drivenr: Byte;var dir: shortstring)`
 `procedure getdir(drivenr: Byte;var dir: ansistring)`

Visibility: default

Description: `Getdir` returns in `dir` the current directory on the drive `drivenr`, where {`drivenr`} is 1 for the first floppy drive, 3 for the first hard disk etc. A value of 0 returns the directory on the current disk. On linux and unix systems, `drivenr` is ignored, as there is only one directory tree.

Errors: An error is returned under dos, if the drive requested isn't ready.

See also: Chdir ([981](#))

Listing: ./refex/ex29.pp

Program Example29;

{ Program to demonstrate the GetDir function. }

Var S : **String**;

begin

GetDir (0,S);

WriteLn ('Current directory is : ',S);

end.

GetHeapStatus

Synopsis: Return the memory manager heap status.

Declaration: `procedure GetHeapStatus(var status: THeapStatus)`

Visibility: default

GetMem

Synopsis: Allocate new memory on the heap

Declaration: `procedure Getmem(var p: pointer;Size: PtrInt)`
`function GetMem(size: PtrInt) : pointer`

Visibility: default

Description: Getmem reserves Size bytes memory on the heap, and returns a pointer to this memory in p. If no more memory is available, nil is returned.

For an example, see Freemem ([1003](#)).

Errors: None.

See also: Freemem ([1003](#)), Dispose ([992](#)), New ([1022](#))

GetMemory

Synopsis: Alias for GetMem ([1005](#))

Declaration: `procedure Getmemory(var p: pointer;Size: PtrInt)`
`function GetMemory(size: PtrInt) : pointer`

Visibility: default

Description: Getmemory is an alias for GetMem ([1005](#)).

See also: GetMem ([1005](#))

GetMemoryManager

Synopsis: Return current memory manager

Declaration: `procedure GetMemoryManager (var MemMgr: TMemoryManager)`

Visibility: default

Description: `GetMemoryManager` stores the current Memory Manager record in `MemMgr`.

For an example, see `\progref`.

Errors: None.

See also: `SetMemoryManager` ([1040](#)), `IsMemoryManagerSet` ([1017](#))

GetProcessID

Declaration: `function GetProcessID : SizeUInt`

Visibility: default

GetThreadID

Declaration: `function GetThreadID : SizeUInt`

Visibility: default

GetVariantManager

Synopsis: Return the current variant manager.

Declaration: `procedure GetVariantManager (var VarMgr: tvariantmanager)`

Visibility: default

Description: `GetVariantManager` returns the current variant manager in `varmgr`.

See also: `IsVariantManagerSet` ([1017](#)), `SetVariantManager` ([1042](#))

get_caller_addr

Synopsis: Return the address of the caller.

Declaration: `function get_caller_addr (framebp: pointer) : pointer`

Visibility: default

Description: `get_caller_frame` returns a pointer to address (the return address) of the caller of the routine which has as frame `framebp`.

See also: `get_frame` ([1007](#)), `get_caller_frame` ([1007](#)), `Dump_Stack` ([993](#))

get_caller_frame

Synopsis: Return the frame pointer of the caller

Declaration: `function get_caller_frame(framebp: pointer) : pointer`

Visibility: default

Description: `get_caller_frame` returns a pointer to the frame of the caller of the routine which has as frame `framebp`.

See also: `get_caller_addr` ([1006](#)), `get_frame` ([1007](#)), `Dump_Stack` ([993](#))

get_frame

Synopsis: Return the current frame

Declaration: `function get_frame : pointer`

Visibility: default

Description: `get_frame` returns a pointer to the current stack frame.

See also: `get_caller_addr` ([1006](#)), `get_caller_frame` ([1007](#))

halt

Synopsis: Stop program execution.

Declaration: `procedure halt(errnum: Byte)`
`procedure halt`

Visibility: default

Description: `Halt` stops program execution and returns control to the calling program. The optional argument `Errnum` specifies an exit value. If omitted, zero is returned.

Errors: None.

See also: `Exit` ([997](#))

Listing: `./refex/ex30.pp`

Program `Example30`;

{ Program to demonstrate the Halt function. }

```
begin
  Writeln ('Before Halt. ');
  Halt (1); { Stop with exit code 1 }
  Writeln ('After Halt doesn't get executed. ');
end.
```

hexStr

Synopsis: Convert integer value to string with hexadecimal representation.

Declaration: `function hexStr(Val: LongInt;cnt: Byte) : shortstring`
`function hexStr(Val: Int64;cnt: Byte) : shortstring`

Visibility: default

Description: HexStr returns a string with the hexadecimal representation of Value. The string has exactly cnt charaters. (i.e. only the cnt rightmost nibbles are taken into account) To have a complete representation of a Longint-type value, 8 nibbles are needed, i.e. cnt=8.

Errors: None.

See also: Str ([1046](#)), Val ([1054](#)), BinStr ([978](#))

Listing: ./refex/ex81.pp

```

Program example81;

{ Program to demonstrate the HexStr function }

Const Value = 45678;

Var I : longint;

begin
  For I:=1 to 10 do
    Writeln (HexStr(Value,I));
end.
```

hi

Synopsis: Return high byte/word of value.

Declaration: `function hi(b: Byte) : Byte`
`function hi(i: Integer) : Byte`
`function hi(w: Word) : Byte`
`function hi(l: LongInt) : Word`
`function hi(l: DWord) : Word`
`function hi(i: Int64) : DWord`
`function hi(q: QWord) : DWord`

Visibility: default

Description: Hi returns the high byte or word from X, depending on the size of X. If the size of X is 4, then the high word is returned. If the size is 2 then the high byte is returned. Hi cannot be invoked on types of size 1, such as byte or char.

Errors: None

See also: Lo ([1019](#))

Listing: ./refex/ex31.pp

```

Program Example31;

{ Program to demonstrate the Hi function. }

var
  L : Longint;
  W : Word;

begin
  L:=1 Shl 16;      { = $10000 }
  W:=1 Shl 8;       { = $100 }
  WriteLn (Hi(L)); { Prints 1 }
  WriteLn (Hi(W)); { Prints 1 }
end.

```

High

Synopsis: Return highest index of open array or enumerated

Declaration: `function High(Arg: TypeOrVariable) : TOrdinal`

Visibility: default

Description: The return value of High depends on it's argument:

- 1.If the argument is an ordinal type, High returns the highest value in the range of the given ordinal type.
- 2.If the argument is an array type or an array type variable then High returns the highest possible value of it's index.
- 3.If the argument is an open array identifier in a function or procedure, then High returns the highest index of the array, as if the array has a zero-based index.

The return type is always the same type as the type of the argument (This can lead to some nasty surprises !).

Errors: None.

See also: Low ([1020](#)), Ord ([1024](#)), Pred ([1027](#)), Succ ([1048](#))

Listing: ./refex/ex80.pp

```

Program example80;

{ Example to demonstrate the High and Low functions. }

Type TEnum = ( North , East , South , West );
          TRange = 14..55;
          TArray = Array [2..10] of Longint;

Function Average (Row : Array of Longint) : Real;

Var I : longint;
      Temp : Real;

begin

```

```

Temp := Row[0];
For I := 1 to High(Row) do
    Temp := Temp + Row[I];
Average := Temp / (High(Row)+1);
end;

Var A : TEnum;
    B : TRange;
    C : TArray;
    I : longint;

begin
    Writeln ('TEnum goes from : ',Ord(Low(TEnum)), ' to ', Ord(high(TEnum)), '.');
    Writeln ('A goes from : ',Ord(Low(A)), ' to ', Ord(high(A)), '.');
    Writeln ('TRange goes from : ',Ord(Low(TRange)), ' to ', Ord(high(TRange)), '.');
    Writeln ('B goes from : ',Ord(Low(B)), ' to ', Ord(high(B)), '.');
    Writeln ('TArray index goes from : ',Ord(Low(TArray)), ' to ', Ord(high(TArray)), '.');
    Writeln ('C index goes from : ',Low(C), ' to ', high(C), '.');
    For I:=Low(C) to High(C) do
        C[I]:=I;
    Writeln ('Average : ',Average(c));
    Write ('Type of return value is always same as type of argument:');
    Writeln (high(high(word)));
end.

```

Inc

Synopsis: Increase value of integer variable

Declaration: `procedure Inc(var X: TOrdinal)`
`procedure Inc(var X: TOrdinal; Increment: TOrdinal)`

Visibility: default

Description: `Inc` increases the value of `X` with `Increment`. If `Increment` isn't specified, then 1 is taken as a default.

Errors: If range checking is on, then a range check can occur, or an overflow error, when an attempt is made to increase `X` over its maximum value.

See also: `Dec` ([990](#))

Listing: `./refex/ex32.pp`

Program Example32;

{ Program to demonstrate the Inc function. }

Const

```

C : Cardinal = 1;
L : Longint  = 1;
I : Integer  = 1;
W : Word     = 1;
B : Byte     = 1;
SI : ShortInt = 1;
CH : Char    = 'A';

```

begin

```

Inc (C);      { C:=2    }
Inc (L,5);    { L:=6    }
Inc (I,-3);   { I:=-2   }
Inc (W,3);    { W:=4    }
Inc (B,100);  { B:=101  }
Inc (SI,-3);  { SI:=-2  }
Inc (CH,1);   { ch:='B' }
end.

```

Include

Synopsis: Include element in set if it was not yet present.

Declaration: `procedure Include(var S: TSetType;E: TSetElement)`

Visibility: default

Description: Include includes `E` in the set `S` if it is not yet part of the set. `E` should be of the same type as the base type of the set `S`.

Thus, the two following statements do the same thing:

```

S:=S+[E];
Include(S,E);

```

For an example, see Exclude ([995](#))

Errors: If the type of the element `E` is not equal to the base type of the set `S`, the compiler will generate an error.

See also: Exclude ([995](#))

IndexByte

Synopsis: Search for a byte in a memory range.

Declaration: `function IndexByte(const buf;len: SizeInt;b: Byte) : SizeInt`

Visibility: default

Description: IndexByte searches the memory at `buf` for maximally `len` positions for the byte `b` and returns it's position if it found one. If `b` is not found then -1 is returned. The position is zero-based.

Errors: `Buf` and `Len` are not checked to see if they are valid values.

See also: IndexChar ([1012](#)), IndexDWord ([1013](#)), IndexWord ([1014](#)), CompareByte ([983](#))

Listing: `./refex/ex105.pp`

Program Example105;

{ Program to demonstrate the IndexByte function. }

Const

```

  ArraySize = 256;
  MaxValue = 256;

```



```

Var
  Buffer : Array[1..ArraySize] of Byte;
  I,J : longint;
  K : Byte;

begin
  Randomize;
  For I:=1 To ArraySize do
    Buffer[I]:=Random(MaxValue);
  For I:=1 to 10 do
    begin
      K:=Random(MaxValue);
      J:=IndexByte(Buffer,ArraySize,K);
      if J=-1 then
        WriteLn('Value ',K,' was not found in buffer.')
      else
        WriteLn('Found ',K,' at position ',J,' in buffer');
      end;
    end.

```

IndexChar

Synopsis: Search for a character in a memory range.

Declaration: `function IndexChar(const buf;len: SizeInt;b: Char) : SizeInt`

Visibility: default

Description: IndexChar searches the memory at buf for maximally len positions for the character b and returns it's position if it found one. If b is not found then -1 is returned. The position is zero-based. The IndexChar0 variant stops looking if a null character is found, and returns -1 in that case.

Errors: Buf and Len are not checked to see if they are valid values.

See also: IndexByte ([1011](#)), IndexDWord ([1013](#)), IndexWord ([1014](#)), CompareChar ([984](#))

Listing: ./refex/ex108.pp

Program Example108;

{ Program to demonstrate the IndexChar function. }

Const

```

  ArraySize = 1000;
  MaxValue = 26;

```

Var

```

  Buffer : Array[1..ArraySize] of Char;
  I,J : longint;
  K : Char;

```

begin

```

  Randomize;
  For I:=1 To ArraySize do
    Buffer[I]:=chr(Ord('A')+Random(MaxValue));
  For I:=1 to 10 do
    begin
      K:=chr(Ord('A')+Random(MaxValue));

```

```

J:=IndexChar( Buffer , ArraySize ,K);
if J=-1 then
  WriteLn('Value ',K,' was not found in buffer. ')
else
  WriteLn('Found ',K,' at position ',J,' in buffer ');
end;
end.

```

IndexChar0

Synopsis: Return index of a character in null-terminated array of char.

Declaration: `function IndexChar0(const buf;len: SizeInt;b: Char) : SizeInt`

Visibility: default

Description: `IndexChar0` returns the index of the character `b` in the null-terminated array `Buf`. At most `len` characters will be searched, or the null character if it is encountered first. If the character is not found, 0 is returned.

Errors: On error, 0 is returned.

See also: `IndexByte` ([1011](#)), `IndexChar` ([1012](#)), `IndexWord` ([1014](#)), `IndexDWord` ([1013](#)), `CompareChar0` ([985](#))

IndexDWord

Synopsis: Search for a DWord value in a memory range.

Declaration: `function IndexDWord(const buf;len: SizeInt;b: DWord) : SizeInt`

Visibility: default

Description: `IndexChar` searches the memory at `buf` for maximally `len` positions for the DWord `DW` and returns it's position if it found one. If `DW` is not found then -1 is returned. The position is zero-based.

Errors: `Buf` and `Len` are not checked to see if they are valid values.

See also: `IndexByte` ([1011](#)), `IndexChar` ([1012](#)), `IndexWord` ([1014](#)), `CompareDWord` ([985](#))

Listing: `./refex/ex106.pp`

Program `Example106;`

{ Program to demonstrate the IndexDWord function. }

Const

```

  ArraySize = 1000;
  MaxValue = 1000;

```

Var

```

  Buffer : Array[1..ArraySize] of DWord;
  I,J : longint;
  K : DWord;

```

begin

```

  Randomize;
  For I:=1 To ArraySize do
    Buffer[I]:=Random(MaxValue);

```

```

For I:=1 to 10 do
  begin
    K:=Random(MaxValue);
    J:=IndexDWord( Buffer , ArraySize ,K);
    if J=-1 then
      Writeln( 'Value ',K, ' was not found in buffer.')
    else
      Writeln( 'Found ',K, ' at position ',J, ' in buffer');
    end;
  end.

```

Indexword

Synopsis: Search for a WORD value in a memory range.

Declaration: `function Indexword(const buf;len: SizeInt;b: Word) : SizeInt`

Visibility: default

Description: IndexChar searches the memory at buf for maximally len positions for the Word W and returns it's position if it found one. If W is not found then -1 is returned.

Errors: Buf and Len are not checked to see if they are valid values.

See also: IndexByte ([1011](#)), IndexDWord ([1013](#)), IndexChar ([1012](#)), CompareWord ([986](#))

Listing: ./refex/ex107.pp

Program Example107;

{ Program to demonstrate the IndexWord function. }

Const

```

  ArraySize = 1000;
  MaxValue = 1000;

```

Var

```

  Buffer : Array[1..ArraySize] of Word;
  I,J : longint;
  K : Word;

```

begin

```

  Randomize;
  For I:=1 To ArraySize do
    Buffer[I]:=Random(MaxValue);
  For I:=1 to 10 do
    begin
      K:=Random(MaxValue);
      J:=IndexWord( Buffer , ArraySize ,K);
      if J=-1 then
        Writeln( 'Value ',K, ' was not found in buffer.')
      else
        Writeln( 'Found ',K, ' at position ',J, ' in buffer');
      end;
    end;

```

end.

Insert

Synopsis: Insert one string in another.

Declaration:

```

procedure Insert(const source: shortstring;var s: shortstring;
                index: SizeInt)
procedure Insert(source: Char;var s: shortstring;index: SizeInt)
procedure Insert(const Source: AnsiString;var S: AnsiString;
                Index: SizeInt)
```

Visibility: default

Description: Insert inserts string Source in string S, at position Index, shifting all characters after Index to the right. The resulting string is truncated at 255 characters, if needed. (i.e. for shortstrings)

Errors: None.

See also: Delete ([991](#)), Copy ([989](#)), Pos ([1026](#))

Listing: ./refex/ex33.pp

Program Example33;

{ Program to demonstrate the Insert function. }

Var S : **String**;

begin

 S:= 'Free Pascal is difficult to use !';

Insert ('NOT ',S,**pos**('difficult ',S));

writeln (s);

end.

int

Synopsis: Calculate integer part of floating point value.

Declaration: function int(d: ValReal) : ValReal

Visibility: default

Description: Int returns the integer part of any Real X, as a Real.

Errors: None.

See also: Frac ([1003](#)), Round ([1035](#))

Listing: ./refex/ex34.pp

Program Example34;

{ Program to demonstrate the Int function. }

begin

Writeln (**Int**(123.456):0:1); { Prints 123.0 }

Writeln (**Int**(-123.456):0:1); { Prints -123.0 }

end.

IOResult

Synopsis: Return result of last file IO operation

Declaration: `function IOResult : Word`

Visibility: `default`

Description: IOResult contains the result of any input/output call, when the `{\$i-}` compiler directive is active, disabling IO checking. When the flag is read, it is reset to zero. If IOResult is zero, the operation completed successfully. If non-zero, an error occurred. The following errors can occur:

dos errors :

- 2**File not found.
- 3**Path not found.
- 4**Too many open files.
- 5**Access denied.
- 6**Invalid file handle.
- 12**Invalid file-access mode.
- 15**Invalid disk number.
- 16**Cannot remove current directory.
- 17**Cannot rename across volumes.

I/O errors :

- 100**Error when reading from disk.
- 101**Error when writing to disk.
- 102**File not assigned.
- 103**File not open.
- 104**File not opened for input.
- 105**File not opened for output.
- 106**Invalid number.

Fatal errors :

- 150**Disk is write protected.
- 151**Unknown device.
- 152**Drive not ready.
- 153**Unknown command.
- 154**CRC check failed.
- 155**Invalid drive specified..
- 156**Seek error on disk.
- 157**Invalid media type.
- 158**Sector not found.
- 159**Printer out of paper.
- 160**Error when writing to device.
- 161**Error when reading from device.
- 162**Hardware failure.

Errors: None.

Listing: ./refex/ex35.pp

Program Example35;

```
{ Program to demonstrate the IOResult function. }
```

Var F : text;

begin

```
  Assign ( f , paramstr(1));
  { $i- }
  Reset ( f );
  { $i+ }
```

If IOresult <> 0 **then**

```
  writeln ( 'File ', paramstr(1), ' doesn't exist' )
else
```

```
  writeln ( 'File ', paramstr(1), ' exists' );
```

end.

IsMemoryManagerSet

Synopsis: Is the memory manager set

Declaration: function IsMemoryManagerSet : Boolean

Visibility: default

Description: IsMemoryManagerSet will return True if the memory manager has been set to another value than the system heap manager, it will return False otherwise.

Errors: None.

See also: SetMemoryManager ([1040](#)), GetMemoryManager ([1006](#))

IsVariantManagerSet

Synopsis: Determine if variant manager is currently set.

Declaration: function IsVariantManagerSet : Boolean

Visibility: default

Description: IsVariantManagerSet determines whether the variant manager was set to a correct variant manager. It returns True if it is, or False if it is not.

The routine checks all variant operation handlers, they should all be set correctly.

See also: SetVariantManager ([1042](#)), GetVariantManager ([1006](#))

length

Synopsis: Calculate length of a string.

Declaration: function Length(s: String) : Byte
 function length(c: Char) : Byte
 function Length(const S: AnsiString) : SizeInt

Visibility: default

Description: Length returns the length of the string S, which is limited to 255 for shortstrings. If the string S is empty, 0 is returned. *Note:* The length of the string S is stored in S[0] for shortstrings only. The Length function should always be used on ansistrings and widestrings.

Errors: None.

See also: Pos ([1026](#))

Listing: ./refex/ex36.pp

Program Example36;

{ Program to demonstrate the Length function. }

```
Var S : String;
    I : Integer;

begin
  S:= '';
  for i:=1 to 10 do
    begin
      S:=S+ '*';
      WriteLn (Length(S):2, ' : ', S);
    end;
  end.
```

Ln

Synopsis: Calculate logarithm

Declaration: function ln(d: ValReal) : ValReal

Visibility: default

Description: Ln returns the natural logarithm of the Real parameter X. X must be positive.

Errors: An run-time error will occur when X is negative.

See also: Exp ([998](#)), Power ([1027](#))

Listing: ./refex/ex37.pp

Program Example37;

{ Program to demonstrate the Ln function. }

```
begin
  WriteLn (Ln(1));      { Prints 0 }
  WriteLn (Ln(Exp(1))); { Prints 1 }
end.
```

lo

Synopsis: Return low byte/word of value.

Declaration: `function lo(B: Byte) : Byte`
`function lo(i: Integer) : Byte`
`function lo(w: Word) : Byte`
`function lo(l: LongInt) : Word`
`function lo(l: DWord) : Word`
`function lo(i: Int64) : DWord`
`function lo(q: QWord) : DWord`

Visibility: default

Description: `Lo` returns the low byte of its argument if this is of type `Integer` or `Word`. It returns the low word of its argument if this is of type `Longint` or `Cardinal`.

Errors: None.

See also: `Ord` ([1024](#)), `Chr` ([982](#)), `Hi` ([1008](#))

Listing: `./refex/ex38.pp`

Program `Example38;`

{ Program to demonstrate the Lo function. }

Var `L : Longint;`
`W : Word;`

begin
`L:=(1 Shl 16) + (1 Shl 4); { $10010 }`
`WriteLn (Lo(L)); { Prints 16 }`
`W:=(1 Shl 8) + (1 Shl 4); { $110 }`
`WriteLn (Lo(W)); { Prints 16 }`
end.

longjmp

Synopsis: Jump to address.

Declaration: `procedure longjmp(var S: jmp_buf; value: LongInt)`

Visibility: default

Description: `LongJump` jumps to the address in the `envjmp_buf`, and restores the registers that were stored in it at the corresponding `SetJump` ([1039](#)) call. In effect, program flow will continue at the `SetJump` call, which will return `value` instead of 0. If a value equal to zero is passed, it will be converted to 1 before passing it on. The call will not return, so it must be used with extreme care. This can be used for error recovery, for instance when a segmentation fault occurred.

For an example, see `SetJump` ([1039](#))

Errors: None.

See also: `SetJump` ([1039](#))

Low

Synopsis: Return lowest index of open array or enumerated

Declaration: `function Low(Arg: TypeOrVariable) : TOrdinal`

Visibility: default

Description: The return value of Low depends on it's argument:

- 1.If the argument is an ordinal type, Low returns the lowest value in the range of the given ordinal type.
- 2.If the argument is an array type or an array type variable then Low returns the lowest possible value of it's index.

The return type is always the same type as the type of the argument.

for an example, see High (1009).

Errors: None.

See also: High (1009), Ord (1024), Pred (1027), Succ (1048)

lowerCase

Synopsis: Return lowercase version of a string.

Declaration: `function lowerCase(const s: shortstring) : shortstring; Overload`
`function lowerCase(c: Char) : Char; Overload`
`function lowercase(const s: ansistring) : ansistring`

Visibility: default

Description: Lowercase returns the lowercase version of its argument C. If its argument is a string, then the complete string is converted to lowercase. The type of the returned value is the same as the type of the argument.

Errors: None.

See also: Upcase (1053)

Listing: ./refex/ex73.pp

Program Example73;

{ Program to demonstrate the Lowercase function. }

Var I : Longint;

begin

For i:=ord('A') **to** ord('Z') **do**

write (lowercase(chr(i)));

WriteLn;

WriteLn (Lowercase('ABCDEFGHIJKLMNOPQRSTUVWXYZ'));

end.

MemSize

Synopsis: Return the size of a memory block.

Declaration: `function MemSize(p: pointer) : PtrInt`

Visibility: default

Description: `MemSize` returns the size of a memory block on the heap.

Errors: Passing an invalid pointer may lead to run-time errors (access violations).

See also: `GetMem` ([1005](#)), `FreeMem` ([1003](#))

mkdir

Synopsis: Create a new directory.

Declaration: `procedure mkdir(const s: String)`

Visibility: default

Description: `Mkdir` creates a new directory `S`.

For an example, see `Rmdir` ([1035](#)).

Errors: Depending on the state of the `\var{\{\$I\}}` switch, a runtime error can be generated if there is an error. In the `\var{\{\$I-\}}` state, use `IOResult` to check for errors.

See also: `Chdir` ([981](#)), `Rmdir` ([1035](#))

Move

Synopsis: Move data from one location in memory to another

Declaration: `procedure Move(const source;var dest;count: SizeInt)`

Visibility: default

Description: `Move` moves `Count` bytes from `Source` to `Dest`.

Errors: If either `Dest` or `Source` is outside the accessible memory for the process, then a run-time error will be generated.

See also: `Fillword` ([1002](#)), `Fillchar` ([1000](#))

Listing: `./refex/ex42.pp`

Program `Example42;`

{ Program to demonstrate the Move function. }

Var `S1,S2 : String [30];`

begin

`S1:= 'Hello World ! ';`

`S2:= 'Bye, bye ! ';`

`Move (S1,S2,Sizeof(S1));`

`WriteLn (S2);`

end.

MoveChar0

Synopsis: Move data till first zero character

Declaration: `procedure MoveChar0(const buf1;var buf2;len: SizeInt)`

Visibility: default

Description: MoveChar0 moves Count bytes from Src to Dest, and stops moving if a zero character is found.

Errors: No checking is done to see if Count stays within the memory allocated to the process.

See also: Move ([1021](#))

Listing: ./refex/ex109.pp

Program Example109;

{ Program to demonstrate the MoveChar0 function. }

Var

Buf1, Buf2 : **Array**[1..80] **of** char;
I : longint;

begin

Randomize;

For I:=1 **to** 80 **do**

Buf1[I]:=chr(Random(16)+Ord('A'));

WriteIn('Original buffer');

writeln(Buf1);

Buf1[Random(80)+1]:=#0;

MoveChar0(Buf1, Buf2, 80);

WriteIn('Randomly zero-terminated Buffer');

WriteIn(Buf2);

end.

New

Synopsis: Dynamically allocate memory for variable

Declaration: `procedure New(var P: Pointer)`

`procedure New(var P: Pointer;Cons: TProcedure)`

Visibility: default

Description: New allocates a new instance of the type pointed to by P, and puts the address in P. If P is an object, then it is possible to specify the name of the constructor with which the instance will be created.

For an example, see Dispose ([992](#)).

Errors: If not enough memory is available, Nil will be returned.

See also: Dispose ([992](#)), Freemem ([1003](#)), Getmem ([1005](#))

OctStr

Synopsis: Convert integer to a string with octal representation.

Declaration: `function OctStr(Val: LongInt;cnt: Byte) : shortstring`
`function OctStr(Val: Int64;cnt: Byte) : shortstring`

Visibility: default

Description: `OctStr` returns a string with the octal representation of `Value`. The string has exactly `cnt` characters.

Errors: None.

See also: `Str` ([1046](#)), `Val` ([1054](#)), `BinStr` ([978](#)), `HexStr` ([1008](#))

Listing: `./refex/ex112.pp`

Program `example112;`

{ Program to demonstrate the OctStr function }

Const `Value = 45678;`

Var `l : longint;`

begin

For `l:=1 to 10 do`

Writeln `(OctStr(Value,l));`

For `l:=1 to 16 do`

Writeln `(OctStr(l,3));`

end.

odd

Synopsis: Is a value odd or even ?

Declaration: `function odd(l: LongInt) : Boolean`
`function odd(l: LongWord) : Boolean`
`function odd(l: Int64) : Boolean`
`function odd(l: QWord) : Boolean`

Visibility: default

Description: `Odd` returns `True` if `X` is odd, or `False` otherwise.

Errors: None.

See also: `Abs` ([972](#)), `Ord` ([1024](#))

Listing: `./refex/ex43.pp`

Program `Example43;`

{ Program to demonstrate the Odd function. }

begin

If `Odd(1) Then`

Writeln `('Everything OK with 1 !');`

```

If Not Odd(2) Then
  WriteLn ( 'Everything OK with 2 !' );
end.

```

Ofs

Synopsis: Return offset of a variable.

Declaration: `function Ofs(var X) : LongInt`

Visibility: default

Description: `Ofs` returns the offset of the address of a variable. This function is only supported for compatibility. In Free Pascal, it returns always the complete address of the variable, since Free Pascal is a 32 bit compiler.

Errors: None.

See also: `DSeg` ([993](#)), `CSeg` ([990](#)), `Seg` ([1038](#)), `Ptr` ([1028](#))

Listing: `./refex/ex44.pp`

Program Example44;

{ Program to demonstrate the Ofs function. }

Var W : Pointer;

begin

 W:= Pointer(**Ofs**(W)); *{ W contains its own offset. }*
end.

Ord

Synopsis: Return ordinal value of an ordinal type.

Declaration: `function Ord(X: TOrdinal) : LongInt`

Visibility: default

Description: `Ord` returns the Ordinal value of a ordinal-type variable X.

Errors: None.

See also: `Chr` ([982](#)), `Succ` ([1048](#)), `Pred` ([1027](#)), `High` ([1009](#)), `Low` ([1020](#))

Listing: `./refex/ex45.pp`

Program Example45;

{ Program to demonstrate the Ord, Pred, Succ functions. }

Type

 TEnum = (Zero , One , Two , Three , Four);

Var

```

X : Longint;
Y : TEnum;

begin
  X:=125;
  Writeln (Ord(X)); { Prints 125 }
  X:=Pred(X);
  Writeln (Ord(X)); { prints 124 }
  Y:= One;
  Writeln (Ord(y)); { Prints 1 }
  Y:=Succ(Y);
  Writeln (Ord(Y)); { Prints 2}
end.

```

Paramcount

Synopsis: Return number of command-line parameters passed to the program.

Declaration: `function Paramcount : LongInt`

Visibility: default

Description: `Paramcount` returns the number of command-line arguments. If no arguments were given to the running program, 0 is returned.

Errors: None.

See also: `Paramstr` ([1025](#))

Listing: `./refex/ex46.pp`

Program Example46;

```

{ Program to demonstrate the ParamCount and ParamStr functions. }
Var
  I : Longint;

begin
  Writeln (paramstr(0), ' : Got ',ParamCount,' command-line parameters: ');
  For i:=1 to ParamCount do
    Writeln (ParamStr (i));
end.

```

ParamStr

Synopsis: Return value of a command-line argument.

Declaration: `function ParamStr(l: LongInt) : String`

Visibility: default

Description: `Paramstr` returns the L-th command-line argument. L must be between 0 and `Paramcount`, these values included. The zeroth argument is the path and file name with which the program was started.

The command-line parameters will be truncated to a length of 255, even though the operating system may support bigger command-lines. The `Objpas` unit (used in `objfpc` or `delphi` mode) define versions of `Paramstr` which return the full-length command-line arguments.

When the complete command-line must be accessed, the `argv` pointer should be used to retrieve the real values of the command-line parameters.

For an example, see `Paramcount` ([1025](#)).

Errors: None.

See also: `Paramcount` ([1025](#))

pi

Synopsis: Return the value of PI.

Declaration: `function pi : ValReal`

Visibility: default

Description: `Pi` returns the value of Pi (3.1415926535897932385).

Errors: None.

See also: `Cos` ([989](#)), `Sin` ([1043](#))

Listing: `./refex/ex47.pp`

Program `Example47;`

{ Program to demonstrate the Pi function. }

```
begin
  WriteLn (Pi);           {3.1415926}
  WriteLn (Sin(Pi));
end.
```

Pos

Synopsis: Search for substring in a string.

Declaration: `function Pos(const substr: shortstring;const s: shortstring) : SizeInt`
`function Pos(C: Char;const s: shortstring) : SizeInt`
`function pos(const substr: shortstring;c: Char) : SizeInt`
`function Pos(const Substr: AnsiString;const Source: AnsiString)`
`: SizeInt`
`function Pos(c: Char;const s: AnsiString) : SizeInt`

Visibility: default

Description: `Pos` returns the index of `Substr` in `S`, if `S` contains `Substr`. In case `Substr` isn't found, 0 is returned. The search is case-sensitive.

Errors: None

See also: `Length` ([1017](#)), `Copy` ([989](#)), `Delete` ([991](#)), `Insert` ([1015](#))

Listing: `./refex/ex48.pp`

Program Example48;

{ Program to demonstrate the Pos function . }

Var

S : **String**;

begin

S:= 'The first space in this sentence is at position : ';

Writeln (S, **pos**(' ', S));

S:= 'The last letter of the alphabet doesn't appear in this sentence ';

If (**Pos** ('Z', S)=0) **and** (**Pos** ('z', S)=0) **then**

Writeln (S);

end.

power

Synopsis: Raise float to integer power

Declaration: `function power(bas: ValReal;expo: ValReal) : ValReal`
`function power(bas: Int64;expo: Int64) : Int64`

Visibility: default

Description: Power returns the value of base to the power expon. Base and expon can be of type Longint, in which case the result will also be a Longint.

The function actually returns `Exp (expon*Ln (base))`

Errors: None.

See also: [Exp \(998\)](#), [Ln \(1018\)](#)

Listing: ./refex/ex78.pp

Program Example78;

{ Program to demonstrate the Power function . }

begin

Writeln (Power(**exp**(1.0), 1.0):8:2); *{ Should print 2.72 }*

end.

Pred

Synopsis: Return previous element for an ordinal type.

Declaration: `function Pred(X: TOrdinal) : TOrdinal`

Visibility: default

Description: Pred returns the element that precedes the element that was passed to it. If it is applied to the first value of the ordinal type, and the program was compiled with range checking on (`\var{\{\$R+\}}`), then a run-time error will be generated.

for an example, see [Ord \(1024\)](#)

Errors: Run-time error 201 is generated when the result is out of range.

See also: Ord ([1024](#)), Pred ([1027](#)), High ([1009](#)), Low ([1020](#))

prefetch

Synopsis: Prefetch a memory location

Declaration: `procedure prefetch(const mem)`

Visibility: default

Description: Prefetch can be used to optimize the CPU behaviour by already loading a memory location. It is mainly used as a hint for those processors that support it.

Errors: None.

ptr

Synopsis: Combine segment and offset to pointer

Declaration: `function ptr(sel: LongInt; off: LongInt) : farpointer`

Visibility: default

Description: Ptr returns a pointer, pointing to the address specified by segment Sel and offset Off.

Remark:

1. In the 32-bit flat-memory model supported by Free Pascal, this function is obsolete.
2. The returned address is simply the offset.

Errors: None.

See also: Addr ([974](#))

Listing: ./refex/ex59.pp

Program Example59;

```
{ Program to demonstrate the Ptr (compability) function.
}
```

```
type pString = ^String;
```

```
Var P : pString;
    S : String;
```

```
begin
  S:= 'Hello , World !';
  P:= pString (Ptr (Seg(S), Longint (Ofs(S))));
  {P now points to S !}
  Writeln (P^);
end.
```

RaiseList

Synopsis: List of currently raised exceptions.

Declaration: `function RaiseList : PExceptObject`

Visibility: default

Description: `RaiseList` returns a pointer to the list of currently raised exceptions (i.e. a pointer to the first exception block).

Errors:

Random

Synopsis: Generate random number

Declaration: `function Random(l: LongInt) : LongInt`
`function Random(l: Int64) : Int64`
`function Random : extended`

Visibility: default

Description: `Random` returns a random number larger or equal to 0 and strictly less than L. If the argument L is omitted, a Real number between 0 and 1 is returned. (0 included, 1 excluded)

Errors: None.

See also: `Randomize` ([1030](#))

Listing: `./refex/ex49.pp`

Program `Example49;`

{ Program to demonstrate the Random and Randomize functions. }

Var `I, Count, guess : Longint;`
`R : Real;`

begin

`Randomize; { This way we generate a new sequence every time`
`the program is run }`

`Count:=0;`

`For i:=1 to 1000 do`

`If Random>0.5 then inc(Count);`

`Writeln ('Generated ', Count, ' numbers > 0.5 ');`

`Writeln ('out of 1000 generated numbers. ');`

`count:=0;`

`For i:=1 to 5 do`

`begin`

`write ('Guess a number between 1 and 5 : ');`

`readln(Guess);`

`If Guess=Random(5)+1 then inc(count);`

`end;`

`Writeln ('You guessed ', Count, ' out of 5 correct. ');`

`end.`

Randomize

Synopsis: Initialize random number generator

Declaration: `procedure Randomize`

Visibility: `default`

Description: `Randomize` initializes the random number generator of Free Pascal, by giving a value to `Randseed`, calculated with the system clock.

For an example, see `Random` ([1029](#)).

Errors: None.

See also: `Random` ([1029](#))

Read

Synopsis: Read from a text file into variable

Declaration: `procedure Read(var F: Text; Args: Arguments)`
`procedure Read(Args: Arguments)`

Visibility: `default`

Description: `Read` reads one or more values from a file `F`, and stores the result in `V1`, `V2`, etc.; If no file `F` is specified, then standard input is read. If `F` is of type `Text`, then the variables `V1`, `V2` etc. must be of type `Char`, `Integer`, `Real`, `String` or `PChar`. If `F` is a typed file, then each of the variables must be of the type specified in the declaration of `F`. Untyped files are not allowed as an argument.

Errors: If no data is available, a run-time error is generated. This behavior can be controlled with the `\var{\{$i\}}` compiler switch.

See also: `Readln` ([1031](#)), `Blockread` ([979](#)), `Write` ([1054](#)), `Blockwrite` ([980](#))

Listing: `./refex/ex50.pp`

Program `Example50`;

{ Program to demonstrate the Read(Ln) function. }

```

Var S : String;
      C : Char;
      F : File of char;

begin
  Assign (F, 'ex50.pp');
  Reset (F);
  C:= 'A';
  Writeln ('The characters before the first space in ex50.pp are : ');
  While not Eof(f) and (C<>' ') do
    Begin
      Read (F,C);
      Write (C);
    end;
  Writeln;
  Close (F);
  Writeln ('Type some words. An empty line ends the program. ');
repeat
```

```

    ReadLn (S);
    until S='';
end.

```

ReadLn

Synopsis: Read from a text file into variable and goto next line

Declaration: `procedure ReadLn(var F: Text; Args: Arguments)`
`procedure ReadLn(Args: Arguments)`

Visibility: default

Description: Read reads one or more values from a file F, and stores the result in V1, V2, etc. After that it goes to the next line in the file. The end of the line is marked by the `LineEnding` character sequence (which is platform dependent). The end-of-line marker is not considered part of the line and is ignored.

If no file F is specified, then standard input is read. The variables V1, V2 etc. must be of type Char, Integer, Real, String or PChar.

For an example, see Read ([1030](#)).

Errors: If no data is available, a run-time error is generated. This behavior can be controlled with the `\var{\{$i\}}` compiler switch.

See also: Read ([1030](#)), Blockread ([979](#)), Write ([1054](#)), Blockwrite ([980](#))

Real2Double

Synopsis: Convert Turbo Pascal style real to double.

Declaration: `function Real2Double(r: real48) : double`

Visibility: default

Description: The `Real2Double` function converts a Turbo Pascal style real (6 bytes long) to a native Free Pascal double type. It can be used e.g. to read old binary TP files with FPC and convert them to Free Pascal binary files.

Note that the assignment operator has been overloaded so a `Real48` type can be assigned directly to a double or extended.

Errors: None.

Listing: `./refex/ex110.pp`

```

program Example110;

{ Program to demonstrate the Real2Double function. }

Var
    i : integer;
    R : Real48;
    D : Double;
    E : Extended;
    F : File of Real48;

```

```

begin
  Assign(F, 'reals.dat');
  Reset(f);
  For I:=1 to 10 do
    begin
      Read(F,R);
      D:=Real2Double(R);
      Writeln('Real ',i,' : ',D);
      D:=R;
      Writeln('Real (direct to double) ',i,' : ',D);
      E:=R;
      Writeln('Real (direct to Extended) ',i,' : ',E);
    end;
  Close(f);
end.

```

ReAllocMem

Synopsis: Re-allocate memory on the heap

Declaration: `function ReAllocMem(var p: pointer;Size: PtrInt) : pointer`

Visibility: default

Description: `ReAllocMem` resizes the memory pointed to by `P` so it has size `Size`. The value of `P` may change during this operation. The contents of the memory pointed to by `P` (if any) will be copied to the new location, but may be truncated if the newly allocated memory block is smaller in size. If a larger block is allocated, only the used memory is initialized, extra memory will not be zeroed out.

Note that `P` may be `nil`, in that case the behaviour of `ReAllocMem` is equivalent to `Getmem`.

See also: [GetMem \(1005\)](#), [FreeMem \(1003\)](#)

ReAllocMemory

Synopsis: Alias for `ReAllocMem` ([1032](#))

Declaration: `function ReAllocMemory(var p: pointer;Size: PtrInt) : pointer`

Visibility: default

Description: `ReAllocMemory` is an alias for `ReAllocMem` ([1032](#)).

See also: `ReAllocMem` ([1032](#))

ReleaseExceptionObject

Synopsis: Decrease the reference count of the current exception object.

Declaration: `procedure ReleaseExceptionObject`

Visibility: default

Description: `ReleaseExceptionObject` decreases the reference count of the current exception object. This should be called whenever a reference to the exception object was obtained via the `AcquireExceptionObject` ([973](#)) call.

Calling this method is only valid within an `except` block.

Errors: If there is no current exception object, a run-time error 231 will occur.

See also: [AcquireExceptionObject \(973\)](#)

Rename

Synopsis: Rename file on disk

Declaration: `procedure Rename(var f: File; const s: String)`
`procedure Rename(var f: File; p: PChar)`
`procedure Rename(var f: File; c: Char)`
`procedure Rename(var t: Text; const s: String)`
`procedure Rename(var t: Text; p: PChar)`
`procedure Rename(var t: Text; c: Char)`

Visibility: default

Description: Rename changes the name of the assigned file F to S. F must be assigned, but not opened.

Errors: Depending on the state of the `\var{\{$I\}}` switch, a runtime error can be generated if there is an error. In the `\var{\{$I-\}}` state, use `IOResult` to check for errors.

See also: [Erase \(995\)](#)

Listing: `./refex/ex77.pp`

Program `Example77;`

```
{ Program to demonstrate the Rename function. }
Var F : Text;

begin
  Assign (F, paramstr(1));
  Rename (F, paramstr(2));
end.
```

Reset

Synopsis: Open file for reading

Declaration: `procedure Reset(var f: File; l: LongInt)`
`procedure Reset(var f: File)`
`procedure Reset(var f: TypedFile)`
`procedure Reset(var t: Text)`

Visibility: default

Description: Reset opens a file F for reading. F can be any file type. If F is a text file, or refers to standard I/O (e.g. ") then it is opened read-only, otherwise it is opened using the mode specified in `filemode`. If F is an untyped file, the record size can be specified in the optional parameter L. A default value of 128 is used. File sharing is not taken into account when calling Reset.

Errors: Depending on the state of the `\var{\{$I\}}` switch, a runtime error can be generated if there is an error. In the `\var{\{$I-\}}` state, use `IOResult` to check for errors.

See also: [Rewrite \(1034\)](#), [Assign \(977\)](#), [Close \(982\)](#), [Append \(975\)](#)

Listing: ./refex/ex51.pp

```

Program Example51;

{ Program to demonstrate the Reset function. }

Function FileExists (Name : String) : boolean;

Var F : File;

begin
  {$i-}
  Assign (F,Name);
  Reset (F);
  {$!+}
  FileExists:=(IoResult=0) and (Name<>' ');
  Close (f);
end;

begin
  If FileExists (Paramstr(1)) then
    Writeln ('File found')
  else
    Writeln ('File NOT found');
end.

```

Rewrite

Synopsis: Open file for writing

Declaration: procedure Rewrite(var f: File; l: LongInt)
 procedure Rewrite(var f: File)
 procedure Rewrite(var f: TypedFile)
 procedure Rewrite(var t: Text)

Visibility: default

Description: Rewrite opens a file F for writing. F can be any file type. If F is an untyped or typed file, then it is opened for reading and writing. If F is an untyped file, the record size can be specified in the optional parameter L. Default a value of 128 is used. if Rewrite finds a file with the same name as F, this file is truncated to length 0. If it doesn't find such a file, a new file is created. Contrary to Turbo Pascal, Free Pascal opens the file with mode fmoutput. If it should be opened in fminout mode, an extra call to Reset ([1033](#)) is needed. File sharing is not taken into account when calling Rewrite.

Errors: Depending on the state of the \var{\{\$I\}} switch, a runtime error can be generated if there is an error. In the \var{\{\$I-\}} state, use IOResult to check for errors.

See also: Reset ([1033](#)), Assign ([977](#)), Close ([982](#)), Flush ([1002](#)), Append ([975](#))

Listing: ./refex/ex52.pp

```

Program Example52;

{ Program to demonstrate the Rewrite function. }

Var F : File;

```

```

    l : longint;

begin
    Assign (F, 'Test.tmp');
    { Create the file . Recordsize is 4 }
    Rewrite (F, Sizeof(l));
    For l:=1 to 10 do
        BlockWrite (F, l, 1);
    close (f);
    { F contains now a binary representation of
      10 longints going from 1 to 10 }
end.

```

rmdir

Synopsis: Remove directory when empty.

Declaration: `procedure rmdir(const s: String)`

Visibility: default

Description: Rmdir removes the directory S.

Errors: Depending on the state of the `\var{\{$I\}}` switch, a runtime error can be generated if there is an error. In the `\var{\{$I-\}}` state, use `IOResult` to check for errors.

See also: Chdir ([981](#)), Mkdir ([1021](#))

Listing: ./refex/ex53.pp

Program Example53;

{ Program to demonstrate the Mkdir and Rmdir functions. }

Const D : **String**[8] = 'TEST.DIR';

Var S : **String**;

```

begin
    Writeln ('Making directory ',D);
    Mkdir (D);
    Writeln ('Changing directory to ',D);
    ChDir (D);
    GetDir (0,S);
    Writeln ('Current Directory is : ',S);
    WRiteln ('Going back');
    ChDir ('..');
    Writeln ('Removing directory ',D);
    Rmdir (D);
end.

```

round

Synopsis: Round floating point value to nearest integer number.

Declaration: `function round(d: ValReal) : Int64`

Visibility: default

Description: Round rounds X to the closest integer, which may be bigger or smaller than X.

Errors: None.

See also: [Frac \(1003\)](#), [Int \(1015\)](#), [Trunc \(1052\)](#)

Listing: ./refex/ex54.pp

Program Example54;

{ Program to demonstrate the Round function. }

```
begin
  WriteLn (Round(1234.56)); { Prints 1235 }
  WriteLn (Round(-1234.56)); { Prints -1235 }
  WriteLn (Round(12.3456)); { Prints 12 }
  WriteLn (Round(-12.3456)); { Prints -12 }
end.
```

RunError

Synopsis: Generate a run-time error.

Declaration: `procedure RunError(w: Word)`
`procedure RunError`

Visibility: default

Description: Runerror stops the execution of the program, and generates a run-time error ErrorCode.

Errors: None.

See also: [Exit \(997\)](#), [Halt \(1007\)](#)

Listing: ./refex/ex55.pp

Program Example55;

{ Program to demonstrate the RunError function. }

```
begin
  { The program will stop and emit a run-error 106 }
  RunError (106);
end.
```

Seek

Synopsis: Set file position

Declaration: `procedure Seek(var f: File; Pos: LongInt)`

Visibility: default

Description: Seek sets the file-pointer for file F to record Nr. Count. The first record in a file has Count=0. F can be any file type, except Text. If F is an untyped file, with no record size specified in [Reset \(1033\)](#) or [Rewrite \(1034\)](#), 128 is assumed.

Errors: Depending on the state of the `\var{\{\$I\}}` switch, a runtime error can be generated if there is an error. In the `\var{\{\$I-\}}` state, use `IOResult` to check for errors.

See also: [Eof \(993\)](#), [SeekEof \(1037\)](#), [SeekEoln \(1038\)](#)

Listing: `./refex/ex56.pp`

Program `Example56;`

```
{ Program to demonstrate the Seek function. }

Var
  F : File;
  I, j : longint;

begin
  { Create a file and fill it with data }
  Assign (F, 'test.tmp');
  Rewrite(F); { Create file }
  Close(f);
  FileMode:=2;
  ReSet (F, Sizeof(i)); { Opened read/write }
  For I:=0 to 10 do
    BlockWrite (F, I, 1);
    { Go Back to the begining of the file }
    Seek(F, 0);
    For I:=0 to 10 do
      begin
        BlockRead (F, J, 1);
        If J<>I then
          WriteLn ( 'Error: expected ', i, ', got ', j);
        end;
      Close (f);
    end.
```

SeekEOF

Synopsis: Set file position to end of file

Declaration: `function SeekEOF(var t: Text) : Boolean`
`function SeekEOF : Boolean`

Visibility: default

Description: `SeekEof` returns `True` is the file-pointer is at the end of the file. It ignores all whitespace. Calling this function has the effect that the file-position is advanced until the first non-whitespace character or the end-of-file marker is reached. If the end-of-file marker is reached, `True` is returned. Otherwise, `False` is returned. If the parameter `F` is omitted, standard `Input` is assumed.

Errors: A run-time error is generated if the file `F` isn't opened.

See also: [Eof \(993\)](#), [SeekEoln \(1038\)](#), [Seek \(1036\)](#)

Listing: `./refex/ex57.pp`

Program `Example57;`

```
{ Program to demonstrate the SeekEof function. }
```

```

Var C : Char;

begin
  { this will print all characters from standard input except
    Whitespace characters. }
  While Not SeekEof do
    begin
      Read (C);
      Write (C);
    end;
end.

```

SeekEOLn

Synopsis: Set file position to end of line

Declaration: `function SeekEOLn(var t: Text) : Boolean`
`function SeekEOLn : Boolean`

Visibility: default

Description: `SeekEoln` returns `True` if the file-pointer is at the end of the current line. It ignores all whitespace. Calling this function has the effect that the file-position is advanced until the first non-whitespace character or the end-of-line marker is reached. If the end-of-line marker is reached, `True` is returned. Otherwise, `False` is returned. The end-of-line marker is defined as `#10`, the LineFeed character. If the parameter `F` is omitted, standard `Input` is assumed.

Errors: A run-time error is generated if the file `F` isn't opened.

See also: `Eof` ([993](#)), `SeekEof` ([1037](#)), `Seek` ([1036](#))

Listing: `./refex/ex58.pp`

Program `Example58`;

```

{ Program to demonstrate the SeekEoln function. }
Var
  C : Char;

begin
  { This will read the first line of standard output and print
    all characters except whitespace. }
  While not SeekEoln do
    Begin
      Read (c);
      Write (c);
    end;
end.

```

Seg

Synopsis: Return segment

Declaration: `function Seg(var X) : LongInt`

Visibility: default

Description: Seg returns the segment of the address of a variable. This function is only supported for compatibility. In Free Pascal, it returns always 0, since Free Pascal is a 32 bit compiler, segments have no meaning.

Errors: None.

See also: DSeg ([993](#)), CSeg ([990](#)), OfS ([1024](#)), Ptr ([1028](#))

Listing: ./refex/ex60.pp

Program Example60;

```
{ Program to demonstrate the Seg function. }
Var
  W : Word;

begin
  W:=Seg(W);  { W contains its own Segment }
end.
```

Setjmp

Synopsis: Save current execution point.

Declaration: function Setjmp(var S: jmp_buf) : LongInt

Visibility: default

Description: SetJmp fills env with the necessary data for a jump back to the point where it was called. It returns zero if called in this way. If the function returns nonzero, then it means that a call to LongJmp ([1019](#)) with env as an argument was made somewhere in the program.

Errors: None.

See also: LongJmp ([1019](#))

Listing: ./refex/ex79.pp

program example79;

```
{ Program to demonstrate the setjmp, longjmp functions }

procedure dojmp(var env : jmp_buf; value : longint);

begin
  value:=2;
  Writeln ( 'Going to jump ! ');
  { This will return to the setjmp call, and return value instead of 0 }
  longjmp(env,value);
end;

var env : jmp_buf;

begin
  if setjmp(env)=0 then
    begin
      writeln ( 'Passed first time.' );
```

```

    dojmp(env,2);
  end
else
  writeln ('Passed second time. ');
end.

```

SetLength

Synopsis: Set length of a string.

Declaration: `procedure SetLength(var S: ShortString; len: SizeInt)`
`procedure SetLength(var S: AnsiString; l: SizeInt)`

Visibility: default

Description: `SetLength` sets the length of the string `S` to `Len`. `S` can be an ansistring, a short string or a widestring. For `ShortStrings`, `Len` can maximally be 255. For `AnsiStrings` it can have any value. For `AnsiString` strings, `SetLength { \em must }` be used to set the length of the string.

Errors: None.

See also: `Length` ([1017](#))

Listing: `./refex/ex85.pp`

Program `Example85;`

{ Program to demonstrate the SetLength function. }

Var `S : String;`

```

begin
  FillChar(S[1],100,#32);
  Setlength(S,100);
  Writeln ('"',S,'"');
end.

```

SetMemoryManager

Synopsis: Set a memory manager

Declaration: `procedure SetMemoryManager(const MemMgr: TMemoryManager)`

Visibility: default

Description: `SetMemoryManager` sets the current memory manager record to `MemMgr`.

For an example, see `\progrf`.

Errors: None.

See also: `GetMemoryManager` ([1006](#)), `IsMemoryManagerSet` ([1017](#))

SetMemoryMutexManager

Synopsis: Procedure to set the mutex manager.

Declaration: `procedure SetMemoryMutexManager(var MutexMgr: TMemoryMutexManager)`

Visibility: default

Description: `SetMemoryMutexManager` sets the mutex manager used by the memory manager to `MutexMgr`. The current mutex manager is returned in `MutexMgr`

Errors: None.

See also: `TMemoryMutexManager` ([966](#)), `SetMemoryManager` ([1040](#))

SetString

Synopsis: Set length of a string and copy buffer.

Declaration: `procedure SetString(var S: Shortstring; Buf: PChar; Len: SizeInt)`
`procedure SetString(var S: AnsiString; Buf: PChar; Len: SizeInt)`

Visibility: default

Description: `SetString` sets the length of the string `S` to `Len` and if `Buf` is non-nil, copies `Len` characters from `Buf` into `S`. `S` can be an ansistring, a short string or a widestring. For `ShortStrings`, `Len` can maximally be 255.

Errors: None.

See also: `SetLength` ([1040](#))

SetTextBuf

Synopsis: Set size of text file internal buffer

Declaration: `procedure SetTextBuf(var f: Text; var Buf)`
`procedure SetTextBuf(var f: Text; var Buf; Size: LongInt)`

Visibility: default

Description: `SetTextBuf` assigns an I/O buffer to a text file. The new buffer is located at `Buf` and is `Size` bytes long. If `Size` is omitted, then `SizeOf(Buf)` is assumed. The standard buffer of any text file is 128 bytes long. For heavy I/O operations this may prove too slow. The `SetTextBuf` procedure allows to set a bigger buffer for the IO of the application, thus reducing the number of system calls, and thus reducing the load on the system resources. The maximum size of the newly assigned buffer is 65355 bytes.

Remark:

- Never assign a new buffer to an opened file. A new buffer can be assigned immediately after a call to `Rewrite` ([1034](#)) `Reset` ([1033](#)) or `Append`, but not after the file was read from/written to. This may cause loss of data. If a new buffer must be assigned after read/write operations have been performed, the file should be flushed first. This will ensure that the current buffer is emptied.
- Take care that the assigned buffer is always valid. If a local variable is assigned as a buffer, then after the program exits the local program block, the buffer will no longer be valid, and stack problems may occur.

Errors: No checking on Size is done.

See also: Assign ([977](#)), Reset ([1033](#)), Rewrite ([1034](#)), Append ([975](#))

Listing: ./refex/ex61.pp

Program Example61;

{ Program to demonstrate the SetTextBuf function. }

Var

Fin, Fout : Text;
Ch : Char;
Bufin, Bufout : **Array**[1..10000] of byte;

begin

Assign (Fin, paramstr(1));
Reset (Fin);
Assign (Fout, paramstr(2));
Rewrite (Fout);
{ This is harmless before IO has begun }
{ Try this program again on a big file ,
after commenting out the following 2
lines and recompiling it. }
SetTextBuf (Fin, Bufin);
SetTextBuf (Fout, Bufout);
While not eof(Fin) **do**
 begin
 Read (Fin, ch);
 write (Fout, ch);
 end;
 Close (Fin);
 Close (Fout);

end.

SetTextLineEnding

Synopsis: Set the end-of-line character for the given text file.

Declaration: procedure SetTextLineEnding(var f: Text; Ending: String)

Visibility: default

Description: SetTextLineEnding sets the end-of-line character for the text file F to Ending. By default, this is the string indicated by DefaultTextLineBreakStyle ([943](#)).

Errors: None.

See also: DefaultTextLineBreakStyle ([943](#)), TTextLineBreakStyle ([967](#))

SetVariantManager

Synopsis: Set the current variant manager.

Declaration: procedure SetVariantManager(const VarMgr: tvariantmanager)

Visibility: default

Description: `SetVariantManager` sets the variant manager to `varmgr`.

See also: `IsVariantManagerSet` ([1017](#)), `GetVariantManager` ([1006](#))

sin

Synopsis: Calculate sine of angle

Declaration: `function sin(d: ValReal) : ValReal`

Visibility: default

Description: `Sin` returns the sine of its argument `X`, where `X` is an angle in radians. If the absolute value of the argument is larger than $\sqrt{2}$, then the result is undefined.

Errors: None.

See also: `Cos` ([989](#)), `Pi` ([1026](#)), `Exp` ([998](#)), `Ln` ([1018](#))

Listing: `./refex/ex62.pp`

Program `Example62;`

{ Program to demonstrate the Sin function. }

```
begin
  Writeln (Sin(Pi):0:1); { Prints 0.0 }
  Writeln (Sin(Pi/2):0:1); { Prints 1.0 }
end.
```

SizeOf

Synopsis: Return size of a variable or type.

Declaration: `function SizeOf(X: TAnyType) : LongInt`

Visibility: default

Description: `SizeOf` returns the size, in bytes, of any variable or type-identifier.

Remark: This isn't really a RTL function. Its result is calculated at compile-time, and hard-coded in the executable.

Errors: None.

See also: `Addr` ([974](#))

Listing: `./refex/ex63.pp`

Program `Example63;`

{ Program to demonstrate the SizeOf function. }

```
Var
  I : Longint;
  S : String [10];

begin
  Writeln (SizeOf(I)); { Prints 4 }
  Writeln (SizeOf(S)); { Prints 11 }
end.
```

Space

Synopsis: Return a string of spaces

Declaration: `function Space(b: Byte) : shortstring`

Visibility: default

Description: `Space` returns a shortstring with length B, consisting of spaces.

See also: `StringOfChar` ([1046](#))

Sptr

Synopsis: Return current stack pointer

Declaration: `function Sptr : Pointer`

Visibility: default

Description: `Sptr` returns the current stack pointer.

Errors: None.

See also: `SSeg` ([1045](#))

Listing: `./refex/ex64.pp`

Program `Example64;`

{ Program to demonstrate the SPtr function. }

Var

`P : Longint;`

begin

`P:=Longint(Sptr); { P Contains now the current stack position. }`

end.

sqr

Synopsis: Calculate the square of a value.

Declaration: `function sqr(l: LongInt) : LongInt`

`function sqr(l: Int64) : Int64`

`function sqr(l: QWord) : QWord`

`function sqr(d: ValReal) : ValReal`

Visibility: default

Description: `Sqr` returns the square of its argument X.

Errors: None.

See also: `Sqrt` ([1045](#)), `Ln` ([1018](#)), `Exp` ([998](#))

Listing: `./refex/ex65.pp`

Program Example65;

```
{ Program to demonstrate the Sqr function. }
Var i : Integer;

begin
  For i:=1 to 10 do
    writeln (Sqr(i):3);
  end.
```

sqrt

Synopsis: Calculate the square root of a value

Declaration: function sqrt(d: ValReal) : ValReal

Visibility: default

Description: Sqrt returns the square root of its argument X, which must be positive.

Errors: If X is negative, then a run-time error is generated.

See also: Sqr ([1044](#)), Ln ([1018](#)), Exp ([998](#))

Listing: ./refex/ex66.pp

Program Example66;

```
{ Program to demonstrate the Sqrt function. }

begin
  Writeln (Sqrt(4):0:3); { Prints 2.000 }
  Writeln (Sqrt(2):0:3); { Prints 1.414 }
end.
```

Sseg

Synopsis: Return stack segment register value.

Declaration: function Sseg : Word

Visibility: default

Description: SSeg returns the Stack Segment. This function is only supported for compatibility reasons, as SPtr returns the correct contents of the stackpointer.

Errors: None.

See also: SPtr ([1044](#))

Listing: ./refex/ex67.pp

Program Example67;

```
{ Program to demonstrate the SSeg function. }
Var W : Longint;
```

```
begin
  W:=SSeg;
end.
```

Str

Synopsis: Convert a numerical value to a string.

Declaration: `procedure Str(var X: TNumericType; var S: String)`

Visibility: default

Description: `Str` returns a string which represents the value of `X`. `X` can be any numerical type. The optional `NumPlaces` and `Decimals` specifiers control the formatting of the string.

Errors: None.

See also: `Val` ([1054](#))

Listing: `./refex/ex68.pp`

Program `Example68;`

```
{ Program to demonstrate the Str function. }
```

```
Var S : String;
```

```
Function IntToStr (I : Longint) : String;
```

```
Var S : String;
```

```
begin
```

```
  Str (I,S);
```

```
  IntToStr:=S;
```

```
end;
```

```
begin
```

```
  S:='*'+IntToStr(-233)+'*';
```

```
  Writeln (S);
```

```
end.
```

StringOfChar

Synopsis: Return a string consisting of 1 character repeated N times.

Declaration: `function StringOfChar(c: Char;l: SizeInt) : AnsiString`

Visibility: default

Description: `StringOfChar` creates a new `String` of length `l` and fills it with the character `c`.

It is equivalent to the following calls:

```
SetLength(StringOfChar,l);
```

```
FillChar(Pointer(StringOfChar)^,Length(StringOfChar),c);
```

Errors: None.

See also: `SetLength` ([1040](#))

Listing: `./refex/ex97.pp`

Program `Example97`;

`{ $H+ }`

`{ Program to demonstrate the StringOfChar function. }`

Var `S : String`;

begin

`S := StringOfChar(' ', 40) + 'Aligned at column 41.';`

`WriteLn(s);`

end.

StringToPPChar

Synopsis: Split string in list of null-terminated strings

Declaration: `function StringToPPChar(var S: AnsiString; ReserveEntries: Integer)`
`: PPChar`
`function StringToPPChar(S: PChar; ReserveEntries: Integer) : PPChar`

Visibility: default

Description: `StringToPPChar` splits the string `S` in words, replacing any whitespace with zero characters. It returns a pointer to an array of `pchars` that point to the first letters of the words in `S`. This array is terminated by a `Nil` pointer.

The function does *not* add a zero character to the end of the string unless it ends on whitespace.

The function reserves memory on the heap to store the array of `PChar`; The caller is responsible for freeing this memory.

This function is only available on certain platforms.

Errors: None.

See also: `ArrayStringToPPchar` ([976](#))

strlen

Synopsis: Length of a null-terminated string.

Declaration: `function strlen(p: PChar) : LongInt`

Visibility: default

Description: Returns the length of the null-terminated string `P`.

Errors: None.

strpas

Synopsis: Convert a null-terminated string to a shortstring.

Declaration: `function strpas(p: PChar) : shortstring`

Visibility: default

Description: Converts a null terminated string in P to a Pascal string, and returns this string. The string is truncated at 255 characters.

Errors: None.

Succ

Synopsis: Return next element of ordinal type.

Declaration: `function Succ(X: TOrdinal) : TOrdinal`

Visibility: default

Description: `Succ` returns the element that succeeds the element that was passed to it. If it is applied to the last value of the ordinal type, and the program was compiled with range checking on (`\var{\{$R+\}}`), then a run-time error will be generated.

for an example, see `Ord` (1024).

Errors: Run-time error 201 is generated when the result is out of range.

See also: `Ord` (1024), `Pred` (1027), `High` (1009), `Low` (1020)

Swap

Synopsis: Swap high and low bytes/words of a variable

Declaration: `function swap(X: Word) : Word`
`function Swap(X: Integer) : Integer`
`function swap(X: LongInt) : LongInt`
`function Swap(X: Cardinal) : Cardinal`
`function Swap(X: QWord) : QWord`
`function swap(X: Int64) : Int64`

Visibility: default

Description: `Swap` swaps the high and low order bytes of X if X is of type `Word` or `Integer`, or swaps the high and low order words of X if X is of type `Longint` or `Cardinal`. The return type is the type of X

Errors: None.

See also: `Lo` (1019), `Hi` (1008)

Listing: `./refex/ex69.pp`

Program Example69;

{ Program to demonstrate the Swap function. }

Var W : Word;
 L : Longint;

begin

```

W:=$1234;
W:=Swap(W);
if W<>$3412 then
  writeln ('Error when swapping word !');
L:=$12345678;
L:=Swap(L);
if L<>$56781234 then
  writeln ('Error when swapping Longint !');
end.

```

SysAllocMem

Synopsis: System memory manager: Allocate memory

Declaration: `function SysAllocMem(size: PtrInt) : Pointer`

Visibility: default

Description: `SysFreeMemSize` is the system memory manager implementation for `AllocMem` ([974](#))

See also: `AllocMem` ([974](#))

SysAssert

Synopsis: Standard Assert failure implementation

Declaration: `procedure SysAssert(const Msg: ShortString; const FName: ShortString;
LineNo: LongInt; ErrorAddr: Pointer)`

Visibility: default

Description: `SysAssert` is the standard implementation of the assertion failed code. It is the default value of the `AssertErrorProc` constant. It will print the assert message `Msg` together with the filename `FName` and line number `LineNo` to standard error output (`StdErr`) and will halt the program with exit code 227. The error address `ErrorAddr` is ignored.

See also: `AssertErrorProc` ([943](#))

SysBackTraceStr

Synopsis: Format an address suitable for inclusion in a backtrace

Declaration: `function SysBackTraceStr(Addr: Pointer) : ShortString`

Visibility: default

Description: `SysBackTraceStr` will create a string representation of the address `Addr`, suitable for inclusion in a stack backtrace.

Errors: None.

SysFreemem

Synopsis: System memory manager free routine.

Declaration: `function SysFreemem(p: pointer) : PtrInt`

Visibility: default

Description: `SysFreeem` is the system memory manager implementation for `FreeMem` ([1003](#))

See also: `FreeMem` ([1003](#))

SysFreememSize

Synopsis: System memory manager free routine.

Declaration: `function SysFreememSize(p: pointer;Size: PtrInt) : PtrInt`

Visibility: default

Description: `SysFreeSize` is the system memory manager implementation for `FreeMem` ([1003](#))

See also: `MemSize` ([1021](#))

SysGetHeapStatus

Synopsis: System implementation of `GetHeapStatus` ([1005](#))

Declaration: `procedure SysGetHeapStatus(var status: THeapStatus)`

Visibility: default

Description: `SysGetHeapStatus` is the system implementation of the `GetHeapStatus` ([1005](#)) call.

See also: `GetHeapStatus` ([1005](#))

SysGetmem

Synopsis: System memory manager memory allocator.

Declaration: `function SysGetmem(Size: PtrInt) : Pointer`

Visibility: default

Description: `SysGetmem` is the system memory manager implementation for `GetMem` ([1005](#))

See also: `GetMem` ([1005](#)), `GetMemory` ([1005](#))

SysInitExceptions

Synopsis: Initialize exceptions.

Declaration: `procedure SysInitExceptions`

Visibility: default

Description: `SysInitExceptions` initializes the exception system. This procedure should never be called directly, it is taken care of by the RTL.

SysInitStdIO

Synopsis: Initialize standard input and output.

Declaration: `procedure SysInitStdIO`

Visibility: default

Description: `SysInitStdIO` initializes the standard input and output files: Output (971), Input (971) and `StdErr` (972). This routine is called by the initialization code of the system unit, there should be no need to call it directly.

SysMemSize

Synopsis: System memory manager: free size.

Declaration: `function SysMemSize(p: pointer) : PtrInt`

Visibility: default

Description: `SysFreeMemSize` is the system memory manager implementation for `MemSize` (1021)

See also: `MemSize` (1021)

SysReAllocMem

Synopsis: System memory manager: Reallocate memory

Declaration: `function SysReAllocMem(var p: pointer; size: PtrInt) : Pointer`

Visibility: default

Description: `SysReallocMem` is a help routine for the system memory manager implementation for `ReAllocMem` (1032).

See also: `ReAllocMem` (1032)

SysResetFPU

Synopsis: Reset the floating point unit.

Declaration: `procedure SysResetFPU`

Visibility: default

Description: `SysResetFPU` resets the floating point unit. There should normally be no need to call this unit; the compiler itself takes care of this.

SysTryResizeMem

Synopsis: System memory manager: attempt to resize memory.

Declaration: `function SysTryResizeMem(var p: pointer; size: PtrInt) : Boolean`

Visibility: default

Description: `SysTryResizeMem` is a help routine for the system memory manager implementation for `ReAllocMem` (1032), `SysReAllocMem` (1051)

See also: `SysReAllocMem` (1051), `ReAllocMem` (1032)

trunc

Synopsis: Truncate a floating point value.

Declaration: `function trunc(d: ValReal) : Int64`

Visibility: default

Description: `Trunc` returns the integer part of `X`, which is always smaller than (or equal to) `X` in absolute value.

Errors: None.

See also: `Frac` ([1003](#)), `Int` ([1015](#)), `Round` ([1035](#))

Listing: `./refex/ex70.pp`

Program `Example70`;

{ Program to demonstrate the Trunc function. }

```
begin
  Writeln (Trunc(123.456)); { Prints 123 }
  Writeln (Trunc(-123.456)); { Prints -123 }
  Writeln (Trunc(12.3456)); { Prints 12 }
  Writeln (Trunc(-12.3456)); { Prints -12 }
end.
```

Truncate

Synopsis: Truncate the file at position

Declaration: `procedure Truncate(var F: File)`

Visibility: default

Description: `Truncate` truncates the (opened) file `F` at the current file position.

Errors: Depending on the state of the `\var{\{$I\}}` switch, a runtime error can be generated if there is an error. In the `\var{\{$I-\}}` state, use `IOResult` to check for errors.

See also: `Append` ([975](#)), `Filepos` ([998](#)), `Seek` ([1036](#))

Listing: `./refex/ex71.pp`

Program `Example71`;

{ Program to demonstrate the Truncate function. }

```
Var F : File of longint;
    I,L : Longint;

begin
  Assign (F, 'test.tmp');
  Rewrite (F);
  For I:=1 to 10 Do
    Write (F,I);
  Writeln ( 'Filesize before Truncate : ',FileSize(F));
  Close (f);
  Reset (F);
```

```

Repeat
  Read (F, I);
Until i=5;
Truncate (F);
WriteLn ('Filesize after Truncate : ', Filesize(F));
Close (f);
end.

```

UniqueString

Synopsis: Make sure reference count of string is 1

Declaration: `procedure UniqueString(var S: AnsiString)`

Visibility: default

Description: `UniqueString` ensures that the `ansistring` `S` has reference count 1. It makes a copy of `S` if this is necessary, and returns the copy in `S`

Errors: None.

upCase

Synopsis: Convert a string to all uppercase.

Declaration: `function upCase(const s: shortstring) : shortstring`
`function upCase(c: Char) : Char`
`function upcase(const s: ansistring) : ansistring`

Visibility: default

Description: `UpCase` returns the uppercase version of its argument `C`. If its argument is a string, then the complete string is converted to uppercase. The type of the returned value is the same as the type of the argument.

Errors: None.

See also: `Lowercase` ([1020](#))

Listing: `./refex/ex72.pp`

Program `Example72`;

{ Program to demonstrate the Upcase function. }

Var `I` : `Longint`;

begin

For `i:=ord('a')` **to** `ord('z')` **do**

write (`upcase(chr(i))`);

WriteLn;

{ This doesn't work in TP, but it does in Free Pascal }

WriteLn (`UpCase('abcdefghijklmnopqrstuvwxyzz')`);

end.

Val

Synopsis: Calculate numerical value of a string.

Declaration: `procedure Val(const S: String; var V: var Code: Word)`

Visibility: default

Description: `Val` converts the value represented in the string `S` to a numerical value, and stores this value in the variable `V`, which can be of type `Longint`, `Real` and `Byte`. If the conversion isn't successful, then the parameter `Code` contains the index of the character in `S` which prevented the conversion. The string `S` is allowed to contain spaces in the beginning. The string `S` can contain a number in decimal, hexadecimal, binary or octal format, as described in the language reference.

Errors: If the conversion doesn't succeed, the value of `Code` indicates the position where the conversion went wrong.

See also: `Str` ([1046](#))

Listing: `./refex/ex74.pp`

Program `Example74;`

```
{ Program to demonstrate the Val function. }
Var I, Code : Integer;

begin
  Val (ParamStr (1), I, Code);
  If Code <> 0 then
    Writeln ('Error at position ', code, ' : ', Paramstr (1)[Code])
  else
    Writeln ('Value : ', I);
end.
```

Write

Synopsis: Write variable to a text file

Declaration: `procedure Write(Args: Arguments)`
`procedure Write(var F: Text; Args: Arguments)`

Visibility: default

Description: `Write` writes the contents of the variables `V1`, `V2` etc. to the file `F`. `F` can be a typed file, or a `Text` file. If `F` is a typed file, then the variables `V1`, `V2` etc. must be of the same type as the type in the declaration of `F`. Untyped files are not allowed. If the parameter `F` is omitted, standard output is assumed. If `F` is of type `Text`, then the necessary conversions are done such that the output of the variables is in human-readable format. This conversion is done for all numerical types. Strings are printed exactly as they are in memory, as well as `PChar` types. The format of the numerical conversions can be influenced through the following modifiers: `OutputVariable : NumChars [: Decimals]` This will print the value of `OutputVariable` with a minimum of `NumChars` characters, from which `Decimals` are reserved for the decimals. If the number cannot be represented with `NumChars` characters, `NumChars` will be increased, until the representation fits. If the representation requires less than `NumChars` characters then the output is filled up with spaces, to the left of the generated string, thus resulting in a right-aligned representation. If no formatting is specified, then the number is written using its natural length, with nothing in front of it if it's positive, and a minus sign if it's negative. Real numbers are, by default, written in scientific notation.

Errors: If an error occurs, a run-time error is generated. This behavior can be controlled with the `\var{\{\$i\}}` switch.

See also: [WriteLn \(1055\)](#), [Read \(1030\)](#), [ReadLn \(1031\)](#), [Blockwrite \(980\)](#)

WriteLn

Synopsis: Write variable to a text file and append newline

Declaration: `procedure Writeln(Args: Arguments)`
`procedure WriteLn(var F: Text;Args: Arguments)`

Visibility: default

Description: `WriteLn` does the same as `Write (1054)` for text files, and emits a Carriage Return - LineFeed character pair after that. If the parameter `F` is omitted, standard output is assumed. If no variables are specified, a Carriage Return - LineFeed character pair is emitted, resulting in a new line in the file `F`.

Remark: Under linux and unix, the Carriage Return character is omitted, as customary in Unix environments.

Errors: If an error occurs, a run-time error is generated. This behavior can be controlled with the `\var{\{\$i\}}` switch.

See also: [Write \(1054\)](#), [Read \(1030\)](#), [ReadLn \(1031\)](#), [Blockwrite \(980\)](#)

Listing: `./refex/ex75.pp`

Program Example75;

{ Program to demonstrate the Write(In) function. }

Var

F : **File of** Longint;
L : Longint;

begin

Write ('This is on the first line ! '); *{ No CR/LF pair! }*
Writeln ('And this too... ');
Writeln ('But this is already on the second line ... ');
Assign (f, 'test.tmp');
Rewrite (f);
For L:=1 **to** 10 **do**
 write (F,L); *{ No writeln allowed here ! }*
Close (f);

end.

28.13 TObject

Description

`TObject` is the parent root class for all classes in Object Pascal. If a class has no parent class explicitly declared, it is dependent on `TObject`. `TObject` introduces class methods that deal with the class' type information, and contains all necessary methods to create an instance at runtime, and to dispatch messages to the correct method (both string and integer messages).

Method overview

Page	Method	Description
1062	<code>AfterConstruction</code>	Method called after the constructor was called.
1062	<code>BeforeDestruction</code>	Method called before the destructor is called.
1059	<code>ClassInfo</code>	Return a pointer to the type information for this class.
1059	<code>ClassName</code>	Return the current class name.
1059	<code>ClassNameIs</code>	Check whether the class name equals the given name.
1060	<code>ClassParent</code>	Return the parent class.
1059	<code>ClassType</code>	Return a "class of" pointer for the current class
1058	<code>CleanupInstance</code>	Finalize the class instance.
1056	<code>Create</code>	<code>TObject</code> Constructor
1057	<code>DefaultHandler</code>	Default handler for integer message handlers.
1062	<code>DefaultHandlerStr</code>	Default handler for string messages.
1056	<code>Destroy</code>	<code>TObject</code> destructor.
1061	<code>Dispatch</code>	Dispatch an integer message
1061	<code>DispatchStr</code>	Dispatch a string message.
1062	<code>FieldAddress</code>	Return the address of a field.
1058	<code>Free</code>	Check for <code>Nil</code> and call destructor.
1057	<code>FreeInstance</code>	Clean up instance and free the memory reserved for the instance.
1060	<code>InheritsFrom</code>	Check whether class is an ancestor.
1058	<code>InitInstance</code>	Initialize a new class instance.
1060	<code>InstanceSize</code>	Return the size of an instance.
1061	<code>MethodAddress</code>	Return the address of a method
1061	<code>MethodName</code>	Return the name of a method.
1057	<code>newInstance</code>	Allocate memory on the heap for a new instance
1057	<code>SafeCallException</code>	Handle exception object
1060	<code>StringMessageTable</code>	Return a pointer to the string message table.

TObject.Create

Synopsis: `TObject` Constructor

Declaration: `constructor Create`

Visibility: `public`

Description: `Create` creates a new instance of `TObject`. Currently it does nothing. It is also not virtual, so there is in principle no need to call it directly.

See also: `TObject.Destroy` ([1056](#))

TObject.Destroy

Synopsis: `TObject` destructor.

Declaration: `destructor Destroy; Virtual`

Visibility: `public`

Description: `Destroy` is the destructor of `TObject`. It will clean up the memory assigned to the instance. Descendent classes should override `destroy` if they want to do additional clean-up. No other destructor should be implemented.

It is bad programming practice to call `Destroy` directly. It is better to call the `Free` ([1058](#)) method, because that one will check first if `Self` is different from `Nil`.

To clean up an instance and reset the refence to the instance, it is best to use the `FreeAndNil` (1149) function.

See also: `TObject.Create` (1056), `TObject.Free` (1058)

TObject.newinstance

Synopsis: Allocate memory on the heap for a new instance

Declaration: `function newInstance : TObject; Virtual`

Visibility: public

Description: `NewInstance` allocates memory on the heap for a new instance of the current class. If the memory was allocated, the class will be initialized by a call to `InitInstance` (1058). The function returns the newly initialized instance.

Errors: If not enough memory is available, a `Nil` pointer may be returned, or an exception may be raised.

See also: `TObject.Create` (1056), `TObject.InitInstance` (1058), `TObject.InstanceSize` (1060), `TObject.FreeInstance` (1057)

TObject.FreeInstance

Synopsis: Clean up instance and free the memory reserved for the instance.

Declaration: `procedure FreeInstance; Virtual`

Visibility: public

Description: `FreeInstance` cleans up an instance of the current class, and releases the heap memory occupied by the class instance.

See also: `TObject.Destroy` (1056), `TObject.InitInstance` (1058), `TObject.NewInstance` (1057)

TObject.SafeCallException

Synopsis: Handle exception object

Declaration: `function SafeCallException(exceptobject: TObject;exceptaddr: pointer)
: LongInt; Virtual`

Visibility: public

Description: `SafeCallException` should be overridden to handle exceptions in a method marked with the `savecall` directive. The implementation in `TObject` simply returns zero.

TObject.DefaultHandler

Synopsis: Default handler for integer message handlers.

Declaration: `procedure DefaultHandler(var message); Virtual`

Visibility: public

Description: `DefaultHandler` is the default handler for messages. If a message has an unknown message ID (i.e. does not appear in the table with integer message handlers), then it will be passed to `DefaultHandler` by the `Dispatch` (1061) method.

Errors:

See also: `TObject.Dispatch` ([1061](#)), `TObject.DefaultHandlerStr` ([1062](#))

TObject.Free

Synopsis: Check for `Nil` and call destructor.

Declaration: `procedure Free`

Visibility: `public`

Description: `Free` will check the `Self` pointer and calls `Destroy` ([1056](#)) if it is different from `Nil`. This is a safer method than calling `Destroy` directly. If a reference to the object must be reset as well (a recommended technique), then the function `FreeAndNil` ([1149](#)) should be called.

Errors: None.

See also: `TObject.Destroy` ([1056](#)), `#rtl.sysutils.freeandnil` ([1149](#))

TObject.InitInstance

Synopsis: Initialize a new class instance.

Declaration: `function InitInstance(instance: pointer) : TObject`

Visibility: `public`

Description: `InitInstance` initializes the memory pointer to by `Instance`. This means that the VMT is initialized, and the interface pointers are set up correctly. The function returns the newly initialized instance.

See also: `TObject.NewInstance` ([1057](#)), `TObject.Create` ([1056](#))

TObject.CleanupInstance

Synopsis: Finalize the class instance.

Declaration: `procedure CleanupInstance`

Visibility: `public`

Description: `CleanUpinstance` finalizes the instance, i.e. takes care of all reference counted objects, by decreasing their reference count by 1, and freeing them if their count reaches zero.

Normally, `CleanupInstance` should never be called, it is called automatically when the object is freed with it's constructor.

Errors: None.

See also: `TObject.Destroy` ([1056](#)), `TObject.Free` ([1058](#)), `TObject.InitInstance` ([1058](#))

TObject.ClassType

Synopsis: Return a "class of" pointer for the current class

Declaration: `function ClassType : TClass`

Visibility: public

Description: `ClassType` returns a `TClass` (964) class type reference for the current class.

See also: `TClass` (964), `TObject.ClassInfo` (1059), `TObject.ClassName` (1059)

TObject.ClassInfo

Synopsis: Return a pointer to the type information for this class.

Declaration: `function ClassInfo : pointer`

Visibility: public

Description: `ClassInfo` returns a pointer to the type information for this class. This pointer can be used in the various type information routines.

TObject.ClassName

Synopsis: Return the current class name.

Declaration: `function ClassName : shortstring`

Visibility: public

Description: `ClassName` returns the class name for the current class, in all-uppercase letters. To check for the class name, use the `ClassNameIs` (1059) class method.

Errors: None.

See also: `TObject.ClassInfo` (1059), `TObject.ClassType` (1059), `TObject.ClassNameIs` (1059)

TObject.ClassNameIs

Synopsis: Check whether the class name equals the given name.

Declaration: `function ClassNameIs(const name: String) : Boolean`

Visibility: public

Description: `ClassNameIs` checks whether `Name` equals the class name. It takes of case sensitivity, i.e. it converts both names to uppercase before comparing.

See also: `TObject.ClassInfo` (1059), `TObject.ClassType` (1059), `TObject.ClassName` (1059)

TObject.ClassParent

Synopsis: Return the parent class.

Declaration: `function ClassParent : TClass`

Visibility: public

Description: `ClassParent` returns the class of the parent class of the current class. This is always different from `Nil`, except for `TObject`.

Errors: None.

See also: `TObject.ClassInfo` ([1059](#)), `TObject.ClassType` ([1059](#)), `TObject.ClassName` ([1059](#))

TObject.InstanceSize

Synopsis: Return the size of an instance.

Declaration: `function InstanceSize : LongInt`

Visibility: public

Description: `InstanceSize` returns the number of bytes an instance takes in memory. This is Just the memory occupied by the class structure, and does not take into account any additional memory that might be allocated by the constructor of the class.

Errors: None.

See also: `TObject.InitInstance` ([1058](#)), `TObject.ClassName` ([1059](#)), `TObject.ClassInfo` ([1059](#)), `TObject.ClassType` ([1059](#))

TObject.InheritsFrom

Synopsis: Chck wether class is an ancestor.

Declaration: `function InheritsFrom(aclass: TClass) : Boolean`

Visibility: public

Description: `InheritsFrom` returns `True` if `AClass` is an ancestor class from the current class, and returns `false` if it is not.

Errors:

See also: `TObject.ClassName` ([1059](#)), `TObject.ClassInfo` ([1059](#)), `TObject.ClassType` ([1059](#)), `TClass` ([964](#))

TObject.StringMessageTable

Synopsis: Return a pointer to the string message table.

Declaration: `function StringMessageTable : pstringmessagetable`

Visibility: public

Description: `StringMessageTable` returns a pointer to the string message table, which can be used to look up methods for dispatching a string message. It is used by the `DispatchStr` ([1061](#)) method.

Errors: If there are no string message handlers, `nil` is returned.

See also: `TObject.DispatchStr` ([1061](#)), `TObject.Dispatch` ([1061](#))

TObject.Dispatch

Synopsis: Dispatch an integer message

Declaration: `procedure Dispatch(var message)`

Visibility: public

Description: `Dispatch` looks in the message handler table for a handler that handles message. The message is identified by the first dword (cardinal) in the message structure.

If no matching message handler is found, the message is passed to the `DefaultHandler` (1057) method, which can be overridden by descendent classes to add custom handling of messages.

See also: `TObject.DispatchStr` (1061), `TObject.DefaultHandler` (1057)

TObject.DispatchStr

Synopsis: Dispatch a string message.

Declaration: `procedure DispatchStr(var message)`

Visibility: public

Description: `DispatchStr` extracts the message identifier from `Message` and checks the message handler table to see if a handler for the message is found, and calls the handler, passing along the message.

If no handler is found, the default `DefaultHandlerStr` (1062) is called.

Errors: None.

See also: `TObject.DefaultHandlerStr` (1062), `TObject.Dispatch` (1061), `TObject.DefaultHandler` (1057)

TObject.MethodAddress

Synopsis: Return the address of a method

Declaration: `function MethodAddress(const name: shortstring) : pointer`

Visibility: public

Description: `MethodAddress` returns the address of a method, searching the method by its name. The `Name` parameter specifies which method should be taken. The search is conducted in a case-insensitive manner.

Errors: If no matching method is found, `Nil` is returned.

See also: `TObject.MethodName` (1061), `TObject.FieldAddress` (1062)

TObject.MethodName

Synopsis: Return the name of a method.

Declaration: `function MethodName(address: pointer) : shortstring`

Visibility: public

Description: `MethodName` searches the VMT for a method with the specified address and returns the name of the method.

Errors: If no method with the matching address is found, an empty string is returned.

See also: `TObject.MethodAddress` (1061), `TObject.FieldAddress` (1062)

TObject.FieldAddress

Synopsis: Return the address of a field.

Declaration: `function FieldAddress(const name: shortstring) : pointer`

Visibility: public

Description: `FieldAddress` returns the address of the field with name `name`. The address is the address of the field in the current class instance.

Errors: If no field with the specified name is found, `Nil` is returned.

See also: `TObject.MethodAddress` ([1061](#)), `TObject.MethodName` ([1061](#))

TObject.AfterConstruction

Synopsis: Method called after the constructor was called.

Declaration: `procedure AfterConstruction; Virtual`

Visibility: public

Description: `AfterConstruction` is a method called after the constructor was called. It does nothing in the implementation of `TObject` and must be overridden by descendent classes to provide specific behaviour that is executed after the constructor has finished executing. (for instance, call an event handler)

Errors: None.

See also: `TObject.BeforeDestruction` ([1062](#)), `TObject.Create` ([1056](#))

TObject.BeforeDestruction

Synopsis: Method called before the destructor is called.

Declaration: `procedure BeforeDestruction; Virtual`

Visibility: public

Description: `BeforeDestruction` is a method called before the destructor is called. It does nothing in the implementation of `TObject` and must be overridden by descendent classes to provide specific behaviour that is executed before the destructor has finished executing. (for instance, call an event handler)

Errors: None.

See also: `TObject.AfterConstruction` ([1062](#)), `TObject.Destroy` ([1056](#)), `TObject.Free` ([1058](#))

TObject.DefaultHandlerStr

Synopsis: Default handler for string messages.

Declaration: `procedure DefaultHandlerStr(var message); Virtual`

Visibility: public

Description: `DefaultHandlerStr` is called for string messages which have no handler associated with them in the string message handler table. The implementation of `DefaultHandlerStr` in `TObject` does nothing and must be overridden by descendent classes to provide specific message handling behaviour.

See also: `TObject.DispatchStr` ([1061](#)), `TObject.Dispatch` ([1061](#)), `TObject.DefaultHandler` ([1057](#))

Chapter 29

Reference for unit 'systhrds'

29.1 Used units

Table 29.1: Used units by unit 'systhrds'

Name	Page
unixtype	1064

29.2 Constants, types and variables

Constants

`DefaultStackSize = 32768`

Types

`PEventState = pointer`

`PRTLCriticalSection = ^TRTLCriticalSection`

`PRTLEvent = pointer`

`TAllocateThreadVarsHandler = procedure`

```
TBasicEventCreateHandler = function(EventAttributes: Pointer;  
    AManualReset: Boolean;  
    InitialState: Boolean;  
    const Name: ansistring)  
    : PEventState
```

```

TBasicEventHandler = procedure(state: PEventState)

TBasicEventWaitForHandler = function(timeout: cardinal;
                                     state: PEventState) : LongInt

TBeginThreadHandler = function(sa: Pointer;stacksize: dword;
                               ThreadFunction: TThreadFunc;p: pointer;
                               creationFlags: dword;
                               var ThreadId: THandle) : DWord

TCriticalSectionHandler = procedure(var cs)

TEndThreadHandler = procedure(ExitCode: DWord)

TGetCurrentThreadIdHandler = function : dword

TInitThreadVarHandler = procedure(var offset: dword;size: dword)

TReleaseThreadVarsHandler = procedure

TRelocateThreadVarHandler = function(offset: dword) : pointer

TRTLCreateEventHandler = function : PRTLEvent

TRTLCriticalSection = pthread_mutex_t

TRTLEventHandler = procedure(AEvent: PRTLEvent)

TRTLEventSyncHandler = procedure(m: trtlmethod;p: tprocedure)

trtlmethod = procedure of object

TThreadFunc = function(parameter: pointer) : puint

TThreadGetPriorityHandler = function(threadHandle: dword) : Integer

TThreadHandler = function(threadHandle: dword) : dword

```

```

TThreadManager = record
  InitManager : function : Boolean;
  DoneManager : function : Boolean;
  BeginThread : TBeginThreadHandler;
  EndThread : TEndThreadHandler;
  SuspendThread : TThreadHandler;
  ResumeThread : TThreadHandler;
  KillThread : TThreadHandler;
  ThreadSwitch : TThreadSwitchHandler;
  WaitForThreadTerminate : TWaitForThreadTerminateHandler;
  ThreadSetPriority : TThreadSetPriorityHandler;
  ThreadGetPriority : TThreadGetPriorityHandler;
  GetCurrentThreadId : TGetCurrentThreadIdHandler;
  InitCriticalSection : TCriticalSectionHandler;
  DoneCriticalSection : TCriticalSectionHandler;
  EnterCriticalSection : TCriticalSectionHandler;
  LeaveCriticalSection : TCriticalSectionHandler;
  InitThreadVar : TInitThreadVarHandler;
  RelocateThreadVar : TRelocateThreadVarHandler;
  AllocateThreadVars : TAllocateThreadVarsHandler;
  ReleaseThreadVars : TReleaseThreadVarsHandler;
  BasicEventCreate : TBasicEventCreateHandler;
  BasicEventDestroy : TBasicEventHandler;
  BasicEventResetEvent : TBasicEventHandler;
  BasicEventSetEvent : TBasicEventHandler;
  BasicEventWaitFor : TBasicEventWaitForHandler;
  RTLEventCreate : TRTLCreateEventHandler;
  RTLEventDestroy : TRTLEventHandler;
  RTLEventSetEvent : TRTLEventHandler;
  RTLEventWaitFor : TRTLEventHandler;
  RTLEventSync : TRTLEventSyncHandler;
end

TThreadSetPriorityHandler = function(threadHandle: dword;Prio: LongInt)
                               : Boolean

TThreadSwitchHandler = procedure

TWaitForThreadTerminateHandler = function(threadHandle: dword;
                                         TimeoutMs: LongInt) : dword

```

29.3 Procedures and functions

BasicEventCreate

Declaration: function BasicEventCreate(EventAttributes: Pointer;
 AManualReset: Boolean;InitialState: Boolean;
 const Name: ansistring) : PEventState

Visibility: default

basiceventdestroy

Declaration: `procedure basiceventdestroy(state: PEventState)`

Visibility: default

basiceventResetEvent

Declaration: `procedure basiceventResetEvent(state: PEventState)`

Visibility: default

basiceventSetEvent

Declaration: `procedure basiceventSetEvent(state: PEventState)`

Visibility: default

basiceventWaitFor

Declaration: `function basiceventWaitFor(Timeout: Cardinal; state: PEventState)
: LongInt`

Visibility: default

BeginThread

Declaration: `function BeginThread(sa: Pointer; stacksize: dword;
ThreadFunction: TThreadFunc; p: pointer;
creationFlags: dword; var ThreadId: THandle) : DWord
function BeginThread(ThreadFunction: TThreadFunc) : DWord
function BeginThread(ThreadFunction: TThreadFunc; p: pointer) : DWord
function BeginThread(ThreadFunction: TThreadFunc; p: pointer;
var ThreadId: THandle) : DWord`

Visibility: default

DoneCriticalSection

Declaration: `procedure DoneCriticalSection(var cs: TRTLCriticalSection)`

Visibility: default

EndThread

Declaration: `procedure EndThread(ExitCode: DWord)
procedure EndThread`

Visibility: default

EnterCriticalSection

Declaration: `procedure EnterCriticalSection(var cs: TRTLCriticalSection)`

Visibility: default

GetCurrentThreadId

Declaration: function GetCurrentThreadId : dword

Visibility: default

GetThreadManager

Declaration: function GetThreadManager(var TM: TThreadManager) : Boolean

Visibility: default

InitCriticalSection

Declaration: procedure InitCriticalSection(var cs: TRTLCriticalSection)

Visibility: default

InitThread

Declaration: procedure InitThread(stklen: cardinal)

Visibility: default

KillThread

Declaration: function KillThread(threadHandle: dword) : dword

Visibility: default

LeaveCriticalSection

Declaration: procedure LeaveCriticalSection(var cs: TRTLCriticalSection)

Visibility: default

ResumeThread

Declaration: function ResumeThread(threadHandle: dword) : dword

Visibility: default

RTLEventCreate

Declaration: function RTLEventCreate : PRTLEvent

Visibility: default

RTLeventdestroy

Declaration: procedure RTLeventdestroy(state: PRTLEvent)

Visibility: default

RTLeventSetEvent

Declaration: `procedure RTLeventSetEvent(state: PRTLEvent)`

Visibility: default

RTLeventsync

Declaration: `procedure RTLeventsync(m: trtlmethod; p: tprocedure)`

Visibility: default

RTLeventWaitFor

Declaration: `procedure RTLeventWaitFor(state: PRTLEvent)`

Visibility: default

SetNoThreadManager

Declaration: `procedure SetNoThreadManager`

Visibility: default

SetThreadManager

Declaration: `function SetThreadManager(const NewTM: TThreadManager;
var OldTM: TThreadManager) : Boolean
function SetThreadManager(const NewTM: TThreadManager) : Boolean`

Visibility: default

SuspendThread

Declaration: `function SuspendThread(threadHandle: dword) : dword`

Visibility: default

ThreadGetPriority

Declaration: `function ThreadGetPriority(threadHandle: dword) : Integer`

Visibility: default

ThreadSetPriority

Declaration: `function ThreadSetPriority(threadHandle: dword; Prio: LongInt) : Boolean`

Visibility: default

ThreadSwitch

Declaration: `procedure ThreadSwitch`

Visibility: `default`

WaitForThreadTerminate

Declaration: `function WaitForThreadTerminate(threadHandle: dword; TimeoutMs: LongInt)
: dword`

Visibility: `default`

Chapter 30

Reference for unit 'sysutils'

30.1 Miscellaneous conversion routines

Functions for various conversions.

Table 30.1:

Name	Description
BCDToInt (1103)	Convert BCD number to integer
CompareMem (1106)	Compare two memory regions
FloatToStrF (1137)	Convert float to formatted string
FloatToStr (1137)	Convert float to string
FloatToText (1139)	Convert float to string
FormatFloat (1147)	Format a floating point value
GetDirs (1151)	Split string in list of directories
IntToHex (1156)	return hexadecimal representation of integer
IntToStr (1157)	return decumal representation of integer
StrToIntDef (1184)	Convert string to integer with default value
StrToInt (1183)	Convert string to integer
StrToFloat (1182)	Convert string to float
TextToFloat (1187)	Convert null-terminated string to float

30.2 Date/time routines

Functions for date and time handling.

30.3 FileName handling routines

Functions for file manipulation.

30.4 File input/output routines

Functions for reading/writing to file.

Table 30.2:

Name	Description
DateTimeToFileDate (1110)	Convert DateTime type to file date
DateTimeToStr (1111)	Construct string representation of DateTime
DateTimeToString (1111)	Construct string representation of DateTime
DateTimeToSystemTime (1112)	Convert DateTime to system time
DateTimeToTimeStamp (1113)	Convert DateTime to timestamp
DateToStr (1113)	Construct string representation of date
Date (1110)	Get current date
DayOfWeek (1114)	Get day of week
DecodeDate (1114)	Decode DateTime to year month and day
DecodeTime (1115)	Decode DateTime to hours, minutes and seconds
EncodeDate (1119)	Encode year, day and month to DateTime
EncodeTime (1119)	Encode hours, minutes and seconds to DateTime
FormatDateTime (1147)	Return string representation of DateTime
IncMonth (1154)	Add 1 to month
IsLeapYear (1158)	Determine if year is leap year
MSecsToTimeStamp (1161)	Convert nr of milliseconds to timestamp
Now (1162)	Get current date and time
StrToDateTime (1181)	Convert string to DateTime
StrToDate (1180)	Convert string to date
StrToTime (1185)	Convert string to time
SystemTimeToDateTime (1186)	Convert system time to datetime
TimeStampToDateTime (1188)	Convert time stamp to DateTime
TimeStampToMSecs (1189)	Convert Timestamp to number of millicseconds
TimeToStr (1189)	return string representation of Time
Time (1188)	Get current tyme

30.5 PChar related functions

Most PChar functions are the same as their counterparts in the STRINGS unit. The following functions are the same :

1. [StrCat \(1168\)](#) : Concatenates two PChar strings.
2. [StrComp \(1168\)](#) : Compares two PChar strings.
3. [StrCopy \(1169\)](#) : Copies a PChar string.
4. [StrECopy \(1170\)](#) : Copies a PChar string and returns a pointer to the terminating null byte.
5. [StrEnd \(1170\)](#) : Returns a pointer to the terminating null byte.
6. [StrIComp \(1171\)](#) : Case insensitive compare of 2 PChar strings.
7. [StrLCat \(1172\)](#) : Appends at most L characters from one PChar to another PChar.
8. [StrLComp \(1173\)](#) : Case sensitive compare of at most L characters of 2 PChar strings.
9. [StrLCopy \(1174\)](#) : Copies at most L characters from one PChar to another.
10. [StrLen \(1174\)](#) : Returns the length (exclusive terminating null byte) of a PChar string.
11. [StrLIComp \(1175\)](#) : Case insensitive compare of at most L characters of 2 PChar strings.

Table 30.3:

Name	Description
AddDisk (1088)	Add sisk to list of disk drives
ChangeFileExt (1106)	Change extension of file name
CreateDir (1109)	Create a directory
DeleteFile (1116)	Delete a file
DiskFree (1117)	Free space on disk
DiskSize (1117)	Total size of disk
ExpandFileName (1122)	Create full file name
ExpandUNCFileName (1122)	Create full UNC file name
ExtractFileDir (1123)	Extract directory part of filename
ExtractFileDrive (1123)	Extract drive part of filename
ExtractFileExt (1124)	Extract extension part of filename
ExtractFileName (1124)	Extract name part of filename
ExtractFilePath (1124)	Extrct path part of filename
ExtractRelativePath (1125)	Construct relative path between two files
FileAge (1125)	Return file age
FileDateToDateTime (1127)	Convert file date to system date
FileExists (1128)	Determine whether a file exists on disk
FileGetAttr (1128)	Get attributes of file
FileGetDate (1129)	Get date of last file modification
FileSearch (1131)	Search for file in path
FileSetAttr (1133)	Get file attributes
FileSetDate (1133)	Get file dates
FindFirst (1134)	Start finding a file
FindNext (1135)	Find next file
GetCurrentDir (1150)	Return current working directory
RemoveDir (1163)	Remove a directory from disk
RenameFile (1164)	Rename a file on disk
SetCurrentDir (1165)	Set current working directory
SetDirSeparators (1166)	Set directory separator characters
FindClose (1134)	Stop searching a file
DoDirSeparators (1118)	Replace directory separator characters

12. StrLower (1176) : Converts a PChar to all lowercase letters.
13. StrMove (1176) : Moves one PChar to another.
14. StrNew (1177) : Makes a copy of a PChar on the heap, and returns a pointer to this copy.
15. StrPos (1178) : Returns the position of one PChar string in another?
16. StrRScan (1179) : returns a pointer to the last occurrence of on PChar string in another one.
17. StrScan (1179) : returns a pointer to the first occurrence of on PChar string in another one.
18. StrUpper (1186) : Converts a PChar to all uppercase letters.

The subsequent functions are different from their counterparts in **STRINGS**, although the same examples can be used.

Table 30.4:

Name	Description
FileCreate (1126)	Create a file and return handle
FileOpen (1130)	Open file and return handle
FileRead (1131)	Read from file
FileSeek (1132)	Set file position
FileTruncate (1133)	Truncate file length
FileWrite (1133)	Write to file
FileClose (1126)	Close file handle

30.6 Date and time formatting characters

Various date and time formatting routines accept a format string, to format the date and or time. The following characters can be used to control the date and time formatting:

c shortdateformat + ' ' + shorttimeformat

d day of month

dd day of month (leading zero)

ddd day of week (abbreviation)

dddd day of week (full)

dddddd shortdateformat

ddddddd longdateformat

m month

mm month (leading zero)

mmm month (abbreviation)

mmmm month (full)

y year (four digits)

yy year (two digits)

yyyy year (with century)

h hour

hh hour (leading zero)

n minute

nn minute (leading zero)

s second

ss second (leading zero)

t shorttimeformat

tt longtimeformat

am/pm use 12 hour clock and display am and pm accordingly

a/p use 12 hour clock and display a and p accordingly

/ insert date separator

: insert time separator

"xx" literal text

'xx' literal text

30.7 Formatting strings

Functions for formatting strings.

Table 30.5:

Name	Description
<code>AdjustLineBreaks</code> (1089)	Convert line breaks to line breaks for system
<code>FormatBuf</code> (1146)	Format a buffer
<code>Format</code> (1141)	Format arguments in string
<code>FmtStr</code> (1140)	Format buffer
<code>QuotedStr</code> (1163)	Quote a string
<code>StrFmt</code> (1171)	Format arguments in a string
<code>StrLFmt</code> (1175)	Format maximum L characters in a string
<code>TrimLeft</code> (1190)	Remove whitespace at the left of a string
<code>TrimRight</code> (1191)	Remove whitespace at the right of a string
<code>Trim</code> (1190)	Remove whitespace at both ends of a string

30.8 String functions

Functions for handling strings.

30.9 Used units

30.10 Overview

This documentation describes the `sysutils` unit. The `sysutils` unit was started by Gertjan Schouten, and completed by Michael Van Canneyt. It aims to be compatible to the Delphi `sysutils` unit, but in contrast with the latter, it is designed to work on multiple platforms. It is implemented on all supported platforms.

30.11 Constants, types and variables

Constants

`ConfigExtension` : `String` = `' .cfg '`

Table 30.6:

Name	Description
AnsiCompareStr (1090)	Compare two strings
AnsiCompareText (1091)	Compare two strings, case insensitive
AnsiExtractQuotedStr (1092)	Removes quotes from string
AnsiLastChar (1092)	Get last character of string
AnsiLowerCase (1093)	Convert string to all-lowercase
AnsiQuotedStr (1094)	Quotes a string
AnsiStrComp (1095)	Compare strings case-sensitive
AnsiStrIComp (1096)	Compare strings case-insensitive
AnsiStrLComp (1098)	Compare L characters of strings case sensitive
AnsiStrLIComp (1098)	Compare L characters of strings case insensitive
AnsiStrLastChar (1097)	Get last character of string
AnsiStrLower (1099)	Convert string to all-lowercase
AnsiStrUpper (1100)	Convert string to all-uppercase
AnsiUpperCase (1101)	Convert string to all-uppercase
AppendStr (1102)	Append 2 strings
AssignStr (1103)	Assign value of strings on heap
CompareStr (1107)	Compare two strings case sensitive
CompareText (1108)	Compare two strings case insensitive
DisposeStr (1118)	Remove string from heap
IsValidIdent (1158)	Is string a valid pascal identifier
LastDelimiter (1159)	Last occurrence of character in a string
LeftStr (1160)	Get first N characters of a string
LoadStr (1160)	Load string from resources
LowerCase (1160)	Convert string to all-lowercase
NewStr (1161)	Allocate new string on heap
RightStr (1164)	Get last N characters of a string
StrAlloc (1167)	Allocate memory for string
StrBufSize (1167)	Reserve memory for a string
StrDispose (1169)	Remove string from heap
StrPas (1177)	Convert PChar to pascal string
StrPCopy (1178)	Copy pascal string
StrPLCopy (1178)	Copy N bytes of pascal string
UpperCase (1193)	Convert string to all-uppercase

`ConfigExtension` is the default extension used by the `GetAppConfigFile (1150)` call. It can be set to any valid extension for the current OS.

`CurrencyDecimals` : Byte = 2

`CurrencyDecimals` is the number of decimals to be used when formatting a currency. It is used by the float formatting routines. This constant is initialized by the initialization routines of the `SysUtils` unit.

`CurrencyFormat` : Byte = 1

`CurrencyFormat` is the default format string for positive currencies. It is used by the float formatting routines. This constant is initialized by the initialization routines of the `SysUtils` unit.

`CurrencyString` : String = '\$'

Table 30.7: Used units by unit 'sysutils'

Name	Page
errors	1071
sysconst	1071
Unix	1071
Unixtype	1071

CurrencyString is the currency symbol for the current locale. It is used by the float formatting routines. This constant is initialized by the initialization routines of the SysUtils unit.

DateDelta = 693594

Days between 1/1/0001 and 12/31/1899

DateSeparator : Char = '-'

DateSeparator is the character used by various date/time conversion routines as the character that separates the day from the month and the month from the year in a date notation. It is used by the date formatting routines. This constant is initialized by the initialization routines of the SysUtils unit.

DecimalSeparator : Char = '.'

DecimalSeparator is used to display the decimal symbol in floating point numbers or currencies. It is used by the float formatting routines. This constant is initialized by the initialization routines of the SysUtils unit.

DefaultTextLineBreakStyle : TTextLineBreakStyle = tlbsLF

Default line break style for the current platform.

DirSeparators : Set of Char = ['/', '\']

DirSeparators is a set of characters which are known directory separator characters on all supported platforms. This set is used by the SetDirSeparators ([1166](#)) call to correct pathnames for the current platform.

DriveDelim = DriveSeparator

DriveDelim refers to the system unit's DriveSeparator constant, it is for Delphi compatibility only.

EmptyStr : String = ''

Empty String Constant

EmptyWideStr : WideString = ''

Empty wide string.

faAnyFile = \$00000003f

Use this attribute in the FindFirst (1134) call to find all matching files.

`faArchive = $00000020`

Attribute of a file, meaning the file has the archive bit set. Used in TSearchRec (1086) and FindFirst (1134)

`faDirectory = $00000010`

Attribute of a file, meaning the file is a directory. Used in TSearchRec (1086) and FindFirst (1134)

`faHidden = $00000002`

Attribute of a file, meaning the file is read-only. Used in TSearchRec (1086) and FindFirst (1134)

`faReadOnly = $00000001`

Attribute of a file, meaning the file is read-only. Used in TSearchRec (1086) and FindFirst (1134)

`faSysFile = $00000004`

Attribute of a file, meaning the file is a system file. Used in TSearchRec (1086) and FindFirst (1134)

`faVolumeId = $00000008`

Attribute of a file, meaning the file contains the volume ID. Used in TSearchRec (1086) and FindFirst (1134)

`filerecnamelength = 255`

`filerecnamelength` describes the length of the FileRec (1082) filename field.

`fmOpenRead = $0000`

`fmOpenRead` is used in the FileOpen (1130) call to open a file in read-only mode.

`fmOpenReadWrite = $0002`

`fmOpenReadWrite` is used in the FileOpen (1130) call to open a file in read-write mode.

`fmOpenWrite = $0001`

`fmOpenWrite` is used in the FileOpen (1130) call to open a file in write-only mode.

`fmShareCompat = $0000`

`fmOpenShareCompat` is used in the FileOpen (1130) call OR-ed together with one of `fmOpenReadWrite` (1078), `fmOpenRead` (1078) or `fmOpenWrite` (1078), to open a file in a sharing modus that is equivalent to sharing implemented in MS-DOS.

`fmShareDenyNone = $0040`

`fmOpenShareExclusive` is used in the FileOpen (1130) call OR-ed together with one of `fmOpenReadWrite` (1078), `fmOpenRead` (1078) or `fmOpenWrite` (1078), to open a file so other processes can read/write the file as well.

`fmShareDenyRead = $0030`

`fmOpenShareExclusive` is used in the `FileOpen` (1130) call OR-ed together with one of `fmOpenReadWrite` (1078), `fmOpenRead` (1078) or `fmOpenWrite` (1078), to open a file so other processes cannot read from it.

`fmShareDenyWrite = $0020`

`fmOpenShareExclusive` is used in the `FileOpen` (1130) call OR-ed together with one of `fmOpenReadWrite` (1078), `fmOpenRead` (1078) or `fmOpenWrite` (1078), to open a file exclusively.

`fmShareExclusive = $0010`

`fmOpenShareExclusive` is used in the `FileOpen` (1130) call OR-ed together with one of `fmOpenReadWrite` (1078), `fmOpenRead` (1078) or `fmOpenWrite` (1078), to open a file exclusively.

`fsFromBeginning = 0`

`fsFromBeginning` is used to indicate in the `FileSeek` (1132) call that a seek operation should be started at the start of the file.

`fsFromCurrent = 1`

`fsFromBeginning` is used to indicate in the `FileSeek` (1132) call that a seek operation should be started at the current position in the file.

`fsFromEnd = 2`

`fsFromBeginning` is used to indicate in the `FileSeek` (1132) call that a seek operation should be started at the last position in the file.

`HexDisplayPrefix : String = '$'`

`HexDisplayPrefix` is used by the formatting routines to indicate that the number which follows the prefix is in Hexadecimal notation.

`HoursPerDay = 24`

Number of hours in a day.

`LeadBytes : Set of Char = []`

`LeadBytes` contains the set of bytes that serve as lead byte in a MBCS string.

`LongDateFormat : String = 'dd" "mmmm" "yyyy'`

`LongDateFormat` contains a template to format a date in a long format. It is used by the date formatting routines. This constant is initialized by the initialization routines of the `SysUtils` unit.

`LongDayNames : Array[1..7] of String = ('Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thu.`

`LongDayNames` is an array with the full names of days. It is used by the date formatting routines. This constant is initialized by the initialization routines of the `SysUtils` unit.

`LongMonthNames : Array[1..12] of String = ('January', 'February', 'March', 'April', 'May`

`LongMonthNames` is an array with the full names of months. It is used by the date formatting routines. This constant is initialized by the initialization routines of the `SysUtils` unit.

`LongTimeFormat` : `String` = `'hh:nn:ss'`

`LongTimeFormat` contains a template to format a time in full notation. It is used by the time formatting routines. This constant is initialized by the initialization routines of the `SysUtils` unit.

`MaxCurrency` : `Currency` = `922337203685477.0000`

Maximum currency value

`MaxDateTime` : `TDateTime` = `2958465.99999`

Maximum `TDateTime` value.

`MinCurrency` : `Currency` = `-922337203685477.0000`

Minimum Currency value

`MinDateTime` : `TDateTime` = `-657434.0`

Minimum `TDateTime` value.

`MinsPerDay` = `HoursPerDay * MinsPerHour`

Number of minutes per day.

`MinsPerHour` = `60`

Number of minutes per hour.

`MonthDays` : `Array[Boolean]` of `TDayTable` = `((31,28,31,30,31,30,31,31,30,31,30,31)`

Array with number of days in the months for leap and non-leap years.

`MSecsPerDay` = `SecsPerDay * MSecsPerSec`

Number of milliseconds per day

`MSecsPerSec` = `1000`

Number of milliseconds per second

`NegCurrFormat` : `Byte` = `5`

`CurrencyFormat` is the default format string for negative currencies. It is used by the float formatting routines. This constant is initialized by the initialization routines of the `SysUtils` unit.

`NullStr` : `PString` = `@EmptyStr`

Pointer to an empty string

`PathDelim` = `DirectorySeparator`

`PathDelim` refers to the system unit's `DirectorySeparator` constant, it is for Delphi compatibility only.

`PathSep = PathSeparator`

`PathSep` refers to the system unit's `PathSeparator` constant, it is for Delphi compatibility only.

`SecsPerDay = MinsPerDay * SecsPerMin`

Number of seconds per day

`SecsPerMin = 60`

Number of seconds per minute

`ShortDateFormat : String = 'd/m/y'`

`ShortDateFormat` contains a template to format a date in a short format. It is used by the date formatting routines. This constant is initialized by the initialization routines of the `SysUtils` unit.

`ShortDayNames : Array[1..7] of String = ('Sun', 'Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat')`

`ShortDayNames` is an array with the abbreviated names of days. It is used by the date formatting routines. This constant is initialized by the initialization routines of the `SysUtils` unit.

`ShortMonthNames : Array[1..12] of String = ('Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec')`

`ShortMonthNames` is an array with the abbreviated names of months. It is used by the date formatting routines. This constant is initialized by the initialization routines of the `SysUtils` unit.

`ShortTimeFormat : String = 'hh:nn'`

`ShortTimeFormat` contains a template to format a time in a short notation. It is used by the time formatting routines. This constant is initialized by the initialization routines of the `SysUtils` unit.

`SwitchChars = ['-']`

The characters in this set will be used by the `FindCmdLineSwitch` (1134) function to determine whether a command-line argument is a switch (an option) or a value. If the first character of an argument is in `SwitchChars`, it will be considered an option or switch.

`SysConfigDir : String = ''`

`SysConfigDir` is the default system configuration directory. It is set at application startup by the `sysutils` initialization routines.

This directory may be returned by the `GetAppConfigDir` (1149) call on some systems.

`TextRecBufSize = 256`

Buffer size of text file record.

`TextRecNameLength = 256`

Length of text file record filename field

```
ThousandSeparator : Char = ','
```

ThousandSeparator is used to separate groups of thousands in floating point numbers or currencies. It is used by the float formatting routines. This constant is initialized by the initialization routines of the SysUtils unit.

```
TimeAMString : String = 'AM'
```

TimeAMString is used to display the AM symbol in the time formatting routines. It is used by the time formatting routines. This constant is initialized by the initialization routines of the SysUtils unit.

```
TimePMString : String = 'PM'
```

TimePMString is used to display the PM symbol in the time formatting routines. It is used by the time formatting routines. This constant is initialized by the initialization routines of the SysUtils unit.

```
TimeSeparator : Char = ':'
```

TimeSeparator is used by the time formatting routines to separate the hours from the minutes and the minutes from the seconds. It is used by the time formatting routines. This constant is initialized by the initialization routines of the SysUtils unit.

```
TwoDigitYearCenturyWindow : Word = 50
```

Window to determine what century 2 digit years are in.

```
UnixDateDelta = 25569
```

Number of days between 1.1.1900 and 1.1.1970

Types

```
EHeapException = EHeapMemoryError
```

EHeapMemoryError is raised when an error occurs in the heap management routines.

```
ExceptClass = Class of Exception
```

ExceptClass is a Exception ([1199](#)) class reference.

```
FileRec = packed record
  Handle : THandle;
  Mode : LongInt;
  RecSize : SizeInt;
  _private : Array[1..3*SizeOf(SizeInt)+5*SizeOf(pointer)] of Byte;
  UserData : Array[1..16] of Byte;
  name : Array[0..filerecnamelength] of Char;
end
```

FileRec describes a untyped file. This record is made available so it can be used to implement drivers for other than the normal file system file records.

```
Int64Rec = packed record
end
```

Int64Rec can be used to extract the parts of a Int64: the high and low cardinal, or a zero-based array of 4 words, or a zero based array of 8 bytes. Note that the meaning of the High and Low parts are different on various CPUs.

```
LongRec = packed record
end
```

LongRec can be used to extract the parts of an long Integer: the high and low word, or the 4 separate bytes as a zero-based array of bytes. Note that the meaning of High and Low parts are different on various CPUs.

```
PByteArray = ^TByteArray
```

Generic pointer to TByteArray ([1083](#)). Use to access memory regions as a byte array.

```
PDayTable = ^TDayTable
```

Pointer to TDayTable type.

```
PString = ^String
```

Pointer to a ansistring

```
PWordarray = ^TWordArray
```

Generic pointer to TWordArray ([1087](#)). Use to access memory regions as a word array.

```
TByteArray = Array[0..32767] of Byte
```

TByteArray is a generic array definition, mostly for use as a base type of the PByteArray ([1083](#)) type.

```
TCaseTranslationTable = Array[0..255] of Char
```

TCaseTranslationTable is the type for a lookup table that can convert 255 ascii characters.

```
TDayTable = Array[1..12] of Word
```

Array of day names.

```
TextBuf = Array[0..TextRecBufSize-1] of Char
```

TextBuf is the type for the default buffer in TextRec ([1084](#))

```
TextRec = packed record
  Handle : THandle;
  Mode : LongInt;
  bufsize : SizeInt;
  LineEnd : TLineEndStr;
```



```

    bufpos : SizeInt;
    bufend : SizeInt;
    bufptr : ^TextBuf;
    openfunc : pointer;
    inoutfunc : pointer;
    flushfunc : pointer;
    closefunc : pointer;
    UserData : Array[1..16] of Byte;
    name : Array[0..textrecnamelength-1] of Char;
    buffer : TextBuf;
end

```

TextRec describes a text file. This record is made available so it can be used to implement drivers for other than the normal file system file records.

To implement a driver, an Assign procedure must be implemented, which fills in the various fields of the record. Most notably, the callback functions must be filled in appropriately. After this, the normal file operations will handle all necessary calls to the various callbacks.

TFilename = String

TFileName is used in the TSearchRec (1086) definition.

TFileRec = FileRec

Alias for FileRec (1082) for Delphi compatibility.

TFloatFormat = (ffGeneral, ffExponent, ffFixed, ffNumber, ffCurrency)

Table 30.8: Enumeration values for type TFloatFormat

Value	Explanation
ffCurrency	Monetary format.
ffExponent	Scientific format.
ffFixed	Fixed point format.
ffGeneral	General number format.
ffNumber	Fixed point format with thousand separator

TFloatFormat is used to determine how a float value should be formatted in the FloatToText (1139) function.

```

TFloatRec = record
    Exponent : Integer;
    Negative : Boolean;
    Digits : Array[0..18] of Char;
end

```

TFloatRec is used to describe a floating point value by the FloatToDecimal (1136) function.

TFloatValue = (fvExtended, fvCurrency, fvSingle, fvReal, fvDouble, fvComp)

Table 30.9: Enumeration values for type TFloatValue

Value	Explanation
fvComp	Comp value
fvCurrency	Currency value
fvDouble	Double value
fvExtended	Extended value
fvReal	Real value
fvSingle	Single value

TFloatValue determines which kind of value should be returned in the (untyped) buffer used by the TextToFloat (1187) function.

TGetAppNameEvent = function : String

This callback type is used by the OnGetApplicationName (1087) to return an alternative application name.

TGetTempDirEvent = function(Global: Boolean) : String

Function prototype for OnGetTempDir (1088) handler.

TGetTempFileEvent = function(const Dir: String;const Prefix: String)
: String

Function prototype for OnGetTempFile (1088) handler.

THandle = System.THandle

THandle refers to the definition of THandle in the system unit, and is provided for backward compatibility only.

TIntegerSet = Set of

TIntegerSet is a generic integer subrange set definition whose size fits in a single integer.

TLineEndStr =

TMbcsByteType = (mbSingleByte,mbLeadByte,mbTrailByte)

Table 30.10: Enumeration values for type TMbcsByteType

Value	Explanation
mbLeadByte	Uses lead-byte
mbSingleByte	Single bytes
mbTrailByte	Uses trailing byte

Type of multi-byte character set.

```
TMethod = packed record
  Code : Pointer;
  Data : Pointer;
end
```

TMethod describes a general method pointer, and is used in Run-Time Type Information handling.

```
TProcedure = procedure
```

TProcedure is a general definition of a procedural callback.

```
TReplaceFlags= Set of (rfReplaceAll,rfIgnoreCase)
```

TReplaceFlags determines the behaviour of the StringReplace (1172) function.

```
TSearchRec = record
  Time : LongInt;
  Size : Int64;
  Attr : LongInt;
  Name : TFilename;
  ExcludeAttr : LongInt;
  FindHandle : Pointer;
  Mode : TMode;
end
```

TSearchRec is a search handle description record. It is initialized by a call to FindFirst (1134) and can be used to do subsequent calls to FindNext (1135). It contains the result of these function calls. It must be used to close the search sequence with a call to FindClose (1134).

Remark: Not all fields of this record should be used. Some of the fields are for internal use only.

```
TSysCharSet = Set of Char
```

Generic set of characters type.

```
TSystemTime = record
  Year : Word;
  Month : Word;
  Day : Word;
  Hour : Word;
  Minute : Word;
  Second : Word;
  MilliSecond : Word;
end
```

The System time structure contains the date/time in a human-understandable format.

```
TTerminateProc = function : Boolean
```

TTerminateProc is the procedural type which should be used when adding exit procedures.

```
TTextLineBreakStyle = (tlbsLF,tlbsCRLF,tlbsCR)
```

Table 30.11: Enumeration values for type `TTextLineBreakStyle`

Value	Explanation
<code>tlbsCR</code>	Carriage-return only
<code>tlbsCRLF</code>	Carriage-return and linefeed characters
<code>tlbsLF</code>	Linefeed only

`TTextLineBreakStyle` describes the style of linebreaks to be used in the `AdjustLineBreaks` (1089) function.

```
TTextRec = TextRec
```

Alias for `TextRec` (1084) for Delphi compatibility.

```
TTimeStamp = record
  Time : Integer;
  Date : Integer;
end
```

`TTimeStamp` contains a timestamp, with the date and time parts specified as separate `TDateTime` values.

```
TWordArray = Array[0..16383] of Word
```

`TWordArray` is a generic array definition, mostly for use as a base type of the `PWordArray` (1083) type.

```
WordRec = packed record
  Lo : Byte;
  Hi : Byte;
end
```

`LongRec` can be used to extract the parts of a word: the high and low byte. Note that the meaning of the High and Low parts are different on various CPUs.

Variables

```
LowerCaseTable : TCaseTranslationTable
```

`LowerCaseTable` is used by the `LowerCase` (1160) routine (and friends) to convert a string to all-lowercase characters. It is filled with the appropriate entries by the `SysUtils` unit initialization routines.

```
OnGetApplicationName : TGetAppNameEvent
```

By default, the configuration file routines `GetAppConfigDir` (1149) and `GetAppConfigFile` (1150) use a default application name to construct a directory or filename. This callback can be used to provide an alternative application name.

Since the result of this callback will be used to construct a filename, care should be taken that the returned name does not contain directory separator characters or characters that cannot appear in a filename.

`OnGetTempDir` : `TGetTempDirEvent`

`OnGetTempDir` can be used to provide custom behaviour for the `GetTempDir` (1153) function. Note that the returned name should have a trailing directory delimiter character.

`OnGetTempFile` : `TGetTempFileEvent`

`OnGetTempDir` can be used to provide custom behaviour for the `GetTempFileName` (1154) function. Note that the values for `Prefix` and `Dir` should be observed.

`OnShowException` : `procedure(Msg: ShortString)`

`OnShowException` is the callback that `ShowException` (1166) uses to display a message in a GUI application. For GUI applications, this variable should always be set. Note that no memory may be available when this callback is called, so the callback should already have all resources it needs, when the callback is set.

`UpperCaseTable` : `TCaseTranslationTable`

`UpperCaseTable` is used by the `UpperCase` (1193) routine (and friends) to convert a string to all-uppercase characters. It is filled with the appropriate entries by the `SysUtils` unit initialization routines.

30.12 Procedures and functions

Abort

Synopsis: Abort program execution.

Declaration: `procedure Abort`

Visibility: default

Description: `Abort` raises an `EAbort` (1194) exception.

See also: `Abort` (1088)

AddDisk

Synopsis: Add a disk to the list of known disks (Unix only)

Declaration: `procedure AddDisk(const path: String)`

Visibility: default

Description: On Linux both the `DiskFree` (1117) and `DiskSize` (1117) functions need a file on the specified drive, since is required for the `statfs` system call.

These filenames are set in `drivestr[0..26]`, and the first 4 have been preset to :

Disk 0 ' . ' default drive - hence current directory is used.

Disk 1 ' /fd0/ . ' floppy drive 1.

Disk 2 ' /fd1/ . ' floppy drive 2.

Disk 3 ' / ' C: equivalent of DOS is the root partition.

Drives 4..26 can be set by your own applications with the `AddDisk` call.

The `AddDisk` call adds `Path` to the names of drive files, and returns the number of the disk that corresponds to this drive. If you add more than 21 drives, the count is wrapped to 4.

Errors: None.

See also: `DiskFree` ([1117](#)), `DiskSize` ([1117](#))

AddTerminateProc

Synopsis: Add a procedure to the exit chain.

Declaration: `procedure AddTerminateProc(TermProc: TTerminateProc)`

Visibility: default

Description: `AddTerminateProc` adds `TermProc` to the list of exit procedures. When the program exits, the list of exit procedures is run over, and all procedures are called one by one, in the reverse order that they were added to the exit chain.

Errors: If no memory is available on the heap, an exception may be raised.

See also: `TTerminateProc` ([1086](#)), `CallTerminateProcs` ([1105](#))

AdjustLineBreaks

Synopsis: Convert possible line-endings to the currently valid line ending.

Declaration: `function AdjustLineBreaks(const S: String) : String`
`function AdjustLineBreaks(const S: String; Style: TTextLineBreakStyle)`
`: String`

Visibility: default

Description: `AdjustLineBreaks` will change all `#13` characters with `#13#10` on Windowsnt and dos. On linux, all `#13#10` character pairs are converted to `#10` and single `#13` characters also.

Errors: None.

See also: `AnsiCompareStr` ([1090](#)), `AnsiCompareText` ([1091](#))

Listing: `./sysutex/ex48.pp`

Program `Example48`;

{ This program demonstrates the AdjustLineBreaks function }

Uses `sysutils`;

Const

`S = 'This is a string'#13'with embedded'#10'linefeed and'+`
`#13'CR characters';`

Begin

`WriteLn (AdjustLineBreaks(S));`

End.

AnsiCompareFileName

Synopsis: Compare 2 filenames.

Declaration: `function AnsiCompareFileName(const S1: Ansistring;const S2: Ansistring)
: SizeInt`

Visibility: default

Description: `AnsiCompareFileName` compares 2 filenames `S1` and `S2`, and returns

< 0 if `S1`<`S2`.

= 0 if `S1`=`S2`.

> 0 if `S1`>`S2`.

The function actually checks `FileNameCaseSensitive` and returns the result of `AnsiCompareStr` (1090) or `AnsiCompareText` (1091) depending on whether `FileNameCaseSensitive` is `True` or `False`

Errors: None.

See also: `AnsiCompareStr` (1090), `AnsiCompareText` (1091), `AnsiLowerCaseFileName` (1094)

AnsiCompareStr

Synopsis: Compare 2 ansistrings, case sensitive, ignoring accents characters.

Declaration: `function AnsiCompareStr(const S1: String;const S2: String) : Integer`

Visibility: default

Description: `AnsiCompareStr` compares two strings and returns the following result:

< 0 if `S1`<`S2`.

0 if `S1`=`S2`.

> 0 if `S1`>`S2`.

The comparison takes into account Ansi characters, i.e. it takes care of strange accented characters. Contrary to `AnsiCompareText` (1091), the comparison is case sensitive.

Errors: None.

See also: `AdjustLineBreaks` (1089), `AnsiCompareText` (1091)

Listing: `./sysutex/ex49.pp`

Program `Example49;`

```
{ This program demonstrates the AnsiCompareStr function }  
{ $H+ }
```

Uses `sysutils;`

Procedure `TestIt (S1,S2 : String);`

Var `R : Longint;`

begin

```

R:=AnsiCompareStr(S1,S2);
Write ( ' ',S1, ' is ');
If R<0 then
  write ( 'less than ')
else If R=0 then
  Write ( 'equal to ')
else
  Write ( 'larger than ');
WriteLn ( ' ',S2, ' ');
end;

Begin
  Testit('One string ', 'One smaller string ');
  Testit('One string ', 'one string ');
  Testit('One string ', 'One string ');
  Testit('One string ', 'One tall string ');
End.

```

AnsiCompareText

Synopsis: Compare 2 ansistrings, case insensitive, ignoring accents characters.

Declaration: `function AnsiCompareText(const S1: String;const S2: String) : Integer`

Visibility: default

Description: `AnsiCompareText` compares two strings and returns the following result:

```

<0if S1<S2.
0if S1=S2.
>0if S1>S2.

```

the comparison takes into account Ansi characters, i.e. it takes care of strange accented characters.
 Contrary to `AnsiCompareStr` ([1090](#)), the comparison is case insensitive.

Errors: None.

See also: `AdjustLineBreaks` ([1089](#)), `AnsiCompareText` ([1091](#))

Listing: `./sysutex/ex50.pp`

Program Example49;

```

{ This program demonstrates the AnsiCompareText function }
{$H+}

```

Uses sysutils;

Procedure TestIt (S1,S2 : **String**);

Var R : Longint;

```

begin
  R:=AnsiCompareText(S1,S2);
  Write ( ' ',S1, ' is ');
  If R<0 then
    write ( 'less than ')

```



```

    else If R=0 then
        Write ( 'equal to ' )
    else
        Write ( 'larger than ' );
        WriteLn ( '""', S2, '"" ');
    end;

Begin
    Testit( 'One string ', 'One smaller string ');
    Testit( 'One string ', 'one string ');
    Testit( 'One string ', 'One string ');
    Testit( 'One string ', 'One tall string ');
End.

```

AnsiExtractQuotedStr

Synopsis: Removes the first quoted string from a string.

Declaration: `function AnsiExtractQuotedStr(var Src: PChar; Quote: Char) : String`

Visibility: default

Description: `AnsiExtractQuotedStr` returns the first quoted string in `Src`, and deletes the result from `Src`.

The resulting string has with `Quote` characters removed from the beginning and end of the string (if they are present), and double `Quote` characters replaced by a single `Quote` characters. As such, it reverses the action of `AnsiQuotedStr` ([1094](#)).

Errors: None.

See also: `AnsiQuotedStr` ([1094](#))

Listing: `./sysutex/ex51.pp`

Program `Example51`;

{ This program demonstrates the AnsiQuotedStr function }

Uses `sysutils`;

Var `S : AnsiString`;

Begin

`S := 'He said "Hello" and walked on';`

`S := AnsiQuotedStr(Pchar(S), '""');`

`WriteLn (S);`

`WriteLn (AnsiExtractQuotedStr(Pchar(S), '""));`

End.

AnsiLastChar

Synopsis: Return a pointer to the last character of a string.

Declaration: `function AnsiLastChar(const S: String) : PChar`

Visibility: default

Description: This function returns a pointer to the last character of S. Since multibyte characters are not yet supported, this is the same as @S[Length(S)].

Errors: None.

See also: [AnsiStrLastChar \(1097\)](#)

Listing: ./sysutex/ex52.pp

Program Example52;

{ This program demonstrates the AnsiLastChar function }

Uses sysutils;

Var S : AnsiString;
L : Longint;

Begin

S:= 'This is an ansistring.';
WriteLn ('Last character of S is : ',AnsiLastChar(S));
L:=Longint(AnsiLastChar(S))-Longint(@S[1])+1;
WriteLn ('Length of S is : ',L);

End.

AnsiLowerCase

Synopsis: Return a lowercase version of a string.

Declaration: function AnsiLowerCase(const s: String) : String

Visibility: default

Description: AnsiLowerCase converts the string S to lowercase characters and returns the resulting string. It takes into account the operating system language settings when doing this, so special characters are converted correctly as well.

Remark: On linux, no language setting is taken in account yet.

Errors: None.

See also: [AnsiUpperCase \(1101\)](#), [AnsiStrLower \(1099\)](#), [AnsiStrUpper \(1100\)](#)

Listing: ./sysutex/ex53.pp

Program Example53;

{ This program demonstrates the AnsiLowerCase function }

Uses sysutils;

Procedure Testit (S : String);

begin

WriteLn (S, ' -> ',AnsiLowerCase(S))
end;

Begin

Testit('AN UPPERCASE STRING');

```

    Testit('Some mixed STring');
    Testit('a lowercase string');
End.

```

AnsiLowerCaseFileName

Synopsis: Convert filename to lowercase.

Declaration: `function AnsiLowerCaseFileName(const s: String) : Ansistring`

Visibility: default

Description: `AnsiLowerCaseFileName` simply returns the result of

```
AnsiLowerCase(S);
```

See also: `AnsiLowerCase` ([1093](#)), `AnsiCompareFileName` ([1090](#)), `AnsiUpperCaseFileName` ([1102](#))

AnsiPos

Synopsis: Return Position of one ansistring in another.

Declaration: `function AnsiPos(const substr: String;const s: String) : SizeInt`

Visibility: default

Description: `AnsiPos` does the same as the standard `Pos` function.

See also: `AnsiStrPos` ([1100](#)), `AnsiStrScan` ([1100](#)), `AnsiStrRScan` ([1100](#))

AnsiQuotedStr

Synopsis: Return a quoted version of a string.

Declaration: `function AnsiQuotedStr(const S: String;Quote: Char) : String`

Visibility: default

Description: `AnsiQuotedString` quotes the string `S` and returns the result. This means that it puts the `Quote` character at both the beginning and end of the string and replaces any occurrence of `Quote` in `S` with 2 `Quote` characters. The action of `AnsiQuotedString` can be reversed by `AnsiExtractQuotedStr` ([1092](#)).

For an example, see `AnsiExtractQuotedStr` ([1092](#))

Errors: None.

See also: `AnsiExtractQuotedStr` ([1092](#))

AnsiSameStr

Synopsis: Checks whether 2 strings are the same (case sensitive)

Declaration: `function AnsiSameStr(const s1: String;const s2: String) : Boolean`

Visibility: default

Description: SameText calls AnsiCompareStr (1090) with S1 and S2 as parameters and returns True if the result of that call is zero, or False otherwise.

Errors: None.

See also: AnsiCompareStr (1090), SameText (1165), AnsiSameText (1095)

AnsiSameText

Synopsis: Checks whether 2 strings are the same (case insensitive)

Declaration: `function AnsiSameText(const s1: String;const s2: String) : Boolean`

Visibility: default

Description: SameText calls AnsiCompareText (1091) with S1 and S2 as parameters and returns True if the result of that call is zero, or False otherwise.

Errors:

See also: AnsiCompareText (1091), SameText (1165), AnsiSameStr (1095)

AnsiStrComp

Synopsis: Compare two null-terminated strings. Case sensitive.

Declaration: `function AnsiStrComp(S1: PChar;S2: PChar) : Integer`

Visibility: default

Description: AnsiStrComp compares 2 PChar strings, and returns the following result:

<0if S1<S2.

0if S1=S2.

>0if S1>S2.

The comparison of the two strings is case-sensitive. The function does not yet take internationalization settings into account.

Errors: None.

See also: AnsiCompareText (1091), AnsiCompareStr (1090)

Listing: ./sysutex/ex54.pp

Program Example54;

{ This program demonstrates the AnsiStrComp function }

Uses sysutils;

```

Procedure TestIt (S1,S2 : Pchar);

Var R : Longint;

begin
  R:=AnsiStrComp(S1,S2);
  Write ( ' ',S1, ' is ' );
  If R<0 then
    write ( 'less than ' )
  else If R=0 then
    Write ( 'equal to ' )
  else
    Write ( 'larger than ' );
  WriteLn ( ' ',S2, ' ');
end;

Begin
  TestIt('One string ', 'One smaller string ');
  TestIt('One string ', 'one string ');
  TestIt('One string ', 'One string ');
  TestIt('One string ', 'One tall string ');
End.

```

AnsiStrlComp

Synopsis: Compare two null-terminated strings. Case insensitive.

Declaration: `function AnsiStrlComp(S1: PChar;S2: PChar) : Integer`

Visibility: default

Description: `AnsiStrlComp` compares 2 `PChar` strings, and returns the following result:

```

<0if S1<S2.
0if S1=S2.
>0if S1>S2.

```

The comparison of the two strings is case-insensitive. The function does not yet take internationalization settings into account.

Errors: None.

See also: `AnsiCompareText` ([1091](#)), `AnsiCompareStr` ([1090](#))

Listing: `./sysutex/ex55.pp`

Program Example55;

```
{ This program demonstrates the AnsiStrlComp function }
```

Uses sysutils;

Procedure TestIt (S1,S2 : Pchar);

Var R : Longint;

begin

```

R:=AnsiStrlComp(S1,S2);
Write ( ' ',S1, ' is ' );
If R<0 then
  write ( 'less than ' )
else If R=0 then
  Write ( 'equal to ' )
else
  Write ( 'larger than ' );
WriteLn ( ' ',S2, ' ' );
end;

Begin
  Testit( 'One string ', 'One smaller string ' );
  Testit( 'One string ', 'one string ' );
  Testit( 'One string ', 'One string ' );
  Testit( 'One string ', 'One tall string ' );
End.

```

AnsiStrLastChar

Synopsis: Return a pointer to the last character of a string.

Declaration: `function AnsiStrLastChar(Str: PChar) : PChar`

Visibility: default

Description: Return a pointer to the last character of the null-terminated string.

Errors: None.

See also: [AnsiCompareText \(1091\)](#), [AnsiCompareStr \(1090\)](#)

Listing: ./sysutex/ex56.pp

```

Program Example56;

{ This program demonstrates the AnsiStrLComp function }

Uses sysutils;

Procedure TestIt (S1,S2 : Pchar; L : longint);

Var R : Longint;

begin
  R:=AnsiStrLComp(S1,S2,L);
  Write ( 'First ',L, ' characters of " ',S1, ' are ' );
  If R<0 then
    write ( 'less than ' )
  else If R=0 then
    Write ( 'equal to ' )
  else
    Write ( 'larger than ' );
  WriteLn ( 'those of " ',S2, ' ' );
end;

Begin
  Testit( 'One string ', 'One smaller string ',255);

```

```

Testit('One string','One String',4);
Testit('One string','1 string',0);
Testit('One string','One string.',9);
End.

```

AnsiStrLComp

Synopsis: Compare a limited number of characters of 2 strings

Declaration: `function AnsiStrLComp(S1: PChar;S2: PChar;MaxLen: cardinal) : Integer`

Visibility: default

Description: `AnsiStrLComp` functions the same as `AnsiStrComp` (1095), but compares at most `MaxLen` characters, if this is less than one of the lengths of the passed strings. If the first `MaxLen` characters in both strings are the same, then zero is returned.

Errors: None.

See also: `AnsiStrComp` (1095), `AnsiStrIComp` (1096), `AnsiStrLComp` (1098)

AnsiStrLIComp

Synopsis: Compares a given number of characters of a string, case insensitive.

Declaration: `function AnsiStrLIComp(S1: PChar;S2: PChar;MaxLen: cardinal) : Integer`

Visibility: default

Description: `AnsiStrLIComp` compares the first `Maxlen` characters of 2 `PChar` strings, `S1` and `S2`, and returns the following result:

```

<0if S1<S2.
0if S1=S2.
>0if S1>S2.

```

The comparison of the two strings is case-insensitive. The function does not yet take internationalization settings into account.

Errors: None.

See also: `AnsiCompareText` (1091), `AnsiCompareStr` (1090)

Listing: `./sysutex/ex57.pp`

Program Example57;

```
{ This program demonstrates the AnsiStrLIComp function }
```

Uses sysutils;

Procedure TestIt (S1,S2 : Pchar; L : longint);

Var R : Longint;

begin

```
  R:=AnsiStrLIComp(S1,S2,L);
```

```

Write ( 'First ',L,' characters of "',S1,'" are ');
If R<0 then
  write ( 'less than ')
else If R=0 then
  Write ( 'equal to ')
else
  Write ( 'larger than ');
WriteLn ( 'those of "',S2,'" ');
end;

Begin
  Testit( 'One string', 'One smaller string',255);
  Testit( 'ONE STRING', 'one String',4);
  Testit( 'One string', '1 STRING',0);
  Testit( 'One STRING', 'one string.',9);
End.

```

AnsiStrLower

Synopsis: Convert a null-terminated string to all-lowercase characters.

Declaration: `function AnsiStrLower(Str: PChar) : PChar`

Visibility: default

Description: `AnsiStrLower` converts the `PChar` `Str` to lowercase characters and returns the resulting `pchar`.

Note that `Str` itself is modified, not a copy, as in the case of `AnsiLowerCase` ([1093](#)). It takes into account the operating system language settings when doing this, so special characters are converted correctly as well.

Remark: On unix, no language setting is taken in account yet.

Errors: None.

See also: `AnsiStrUpper` ([1100](#)), `AnsiLowerCase` ([1093](#))

Listing: `./sysutex/ex59.pp`

Program Example59;

{ This program demonstrates the AnsiStrLower function }

Uses sysutils;

Procedure Testit (S : Pchar);

begin

 WriteLn (S, ' -> ',AnsiStrLower(S))

end;

Begin

 Testit('AN UPPERCASE STRING');

 Testit('Some mixed STring');

 Testit('a lowercase string');

End.

AnsiStrPos

Synopsis: Return position of one null-terminated substring in another

Declaration: `function AnsiStrPos(str: PChar; substr: PChar) : PChar`

Visibility: default

Description: `AnsiStrPos` returns a pointer to the first occurrence of `SubStr` in `Str`. If `SubStr` does not occur in `Str` then `Nil` is returned.

Errors: An access violation may occur if either `Str` or `SubStr` point to invalid memory.

See also: `AnsiPos` ([1094](#)), `AnsiStrScan` ([1100](#)), `AnsiStrRScan` ([1100](#))

AnsiStrRScan

Synopsis: Find last occurrence of a character in a null-terminated string.

Declaration: `function AnsiStrRScan(Str: PChar; Chr: Char) : PChar`

Visibility: default

Description: `AnsiStrRScan` returns a pointer to the *last* occurrence of the character `Chr` in `Str`. If `Chr` does not occur in `Str` then `Nil` is returned.

Errors: An access violation may occur if `Str` points to invalid memory.

See also: `AnsiPos` ([1094](#)), `AnsiStrScan` ([1100](#)), `AnsiStrPos` ([1100](#))

AnsiStrScan

Synopsis: Find first occurrence of a character in a null-terminated string.

Declaration: `function AnsiStrScan(Str: PChar; Chr: Char) : PChar`

Visibility: default

Description: `AnsiStrScan` returns a pointer to the *first* occurrence of the character `Chr` in `Str`. If `Chr` does not occur in `Str` then `Nil` is returned.

Errors: An access violation may occur if `Str` points to invalid memory.

See also: `AnsiPos` ([1094](#)), `AnsiStrScan` ([1100](#)), `AnsiStrPos` ([1100](#))

AnsiStrUpper

Synopsis: Convert a null-terminated string to all-uppercase characters.

Declaration: `function AnsiStrUpper(Str: PChar) : PChar`

Visibility: default

Description: `AnsiStrUpper` converts the `PCharStr` to uppercase characters and returns the resulting string. Note that `Str` itself is modified, not a copy, as in the case of `AnsiUpperCase` ([1101](#)). It takes into account the operating system language settings when doing this, so special characters are converted correctly as well.

Remark: On linux, no language setting is taken in account yet.

Errors: None.

See also: [AnsiUpperCase \(1101\)](#), [AnsiStrLower \(1099\)](#), [AnsiLowerCase \(1093\)](#)

Listing: ./sysutex/ex60.pp

Program Example60;

{ This program demonstrates the AnsiStrUpper function }

Uses sysutils;

Procedure Testit (S : Pchar);

begin

 WriteLn (S, ' -> ', AnsiStrUpper(S))

end;

Begin

 Testit('AN UPPERCASE STRING');

 Testit('Some mixed STring');

 Testit('a lowercase string');

End.

AnsiUpperCase

Synopsis: Return an uppercase version of a string, taking into account special characters.

Declaration: function AnsiUpperCase(const s: String) : String

Visibility: default

Description: `AnsiUpperCase` converts the string `S` to uppercase characters and returns the resulting string. It takes into account the operating system language settings when doing this, so special characters are converted correctly as well.

Remark: On linux, no language setting is taken in account yet.

Errors: None.

See also: [AnsiStrUpper \(1100\)](#), [AnsiStrLower \(1099\)](#), [AnsiLowerCase \(1093\)](#)

Listing: ./sysutex/ex61.pp

Program Example60;

{ This program demonstrates the AnsiUpperCase function }

Uses sysutils;

Procedure Testit (S : String);

begin

 WriteLn (S, ' -> ', AnsiUpperCase(S))

end;

Begin

 Testit('AN UPPERCASE STRING');

```

    Testit('Some mixed STring');
    Testit('a lowercase string');
End.

```

AnsiUpperCaseFileName

Synopsis: Convert filename to uppercase.

Declaration: `function AnsiUpperCaseFileName(const s: String) : Ansistring`

Visibility: default

Description: `AnsiUpperCaseFileName` simply returns the result of

```
AnsiUpperCase(S);
```

See also: `AnsiUpperCase` ([1101](#)), `AnsiCompareFileName` ([1090](#)), `AnsiLowerCaseFileName` ([1094](#))

AppendStr

Synopsis: Append one ansistring to another.

Declaration: `procedure AppendStr(var Dest: String; const S: String)`

Visibility: default

Description: `AppendStr` appends `S` to `Dest`.

This function is provided for Delphi compatibility only, since it is completely equivalent to `Dest := Dest + S`.

Errors: None.

See also: `AssignStr` ([1103](#)), `NewStr` ([1161](#)), `DisposeStr` ([1118](#))

Listing: `./sysutex/ex62.pp`

Program Example62;

{ This program demonstrates the AppendStr function }

Uses sysutils;

Var S : AnsiString;

Begin

 S := 'This is an ';

AppendStr(S, 'AnsiString');

Writeln ('S = "', S, '"');

End.

ApplicationName

Synopsis: Return a default application name

Declaration: `function ApplicationName : String`

Visibility: default

Description: `ApplicationName` returns the name of the current application. Standard this is equal to the result of `ParamStr(0)`, but it can be customized by setting the `OnGetApplicationName` ([1087](#)) callback.

Errors: None.

See also: `GetAppConfigDir` ([1149](#)), `OnGetApplicationName` ([1087](#)), `GetAppConfigFile` ([1150](#)), `ConfigExtension` ([1076](#))

AssignStr

Synopsis: Assigns an ansistring to a null-terminated string.

Declaration: `procedure AssignStr(var P: PString; const S: String)`

Visibility: default

Description: `AssignStr` allocates `S` to `P`. The old value of `P` is disposed of.

This function is provided for Delphi compatibility only. `AnsiStrings` are managed on the heap and should be preferred to the mechanism of dynamically allocated strings.

Errors: None.

See also: `NewStr` ([1161](#)), `AppendStr` ([1102](#)), `DisposeStr` ([1118](#))

Listing: `./sysutex/ex63.pp`

Program Example63;

{ This program demonstrates the AssignStr function }
{ \$H+ }

Uses sysutils;

Var P : PString;

Begin

P:=**NewStr**('A first AnsiString');
WriteLn ('Before: P = "', P^, '"');
AssignStr(P, 'A Second ansistring');
WriteLn ('After : P = "', P^, '"');
DisposeStr(P);

End.

BCDToInt

Synopsis: Convert a BCD coded integer to a normal integer.

Declaration: `function BCDToInt(Value: Integer) : Integer`

Visibility: default

Description: `BCDToInt` converts a BCD coded integer to a normal integer.

Errors: None.

See also: `StrToInt` ([1183](#)), `IntToStr` ([1157](#))

Listing: `./sysutex/ex64.pp`

Program `Example64`;

{ This program demonstrates the BCDToInt function }

Uses `sysutils`;

Procedure `Testit` (`L` : `longint`);
begin
 WriteLn (`L`, ' -> ', `BCDToInt(L)`);
end;

Begin
 `Testit(10)`;
 `Testit(100)`;
 `Testit(1000)`;
End.

Beep

Synopsis: Sound the system bell.

Declaration: `procedure Beep`

Visibility: `default`

Description: `Beep` sounds the system bell, if one is available.

Errors: This routine may not be implemented on all platforms.

BoolToStr

Synopsis: Convert a boolean value to a string.

Declaration: `function BoolToStr(B: Boolean) : String`

Visibility: `default`

Description: `BoolToStr` converts the boolean `B` to one of the strings `'TRUE'` or `'FALSE'`

Errors: None.

See also: `StrToBool` ([1179](#))

ByteToCharIndex

Synopsis: Convert a character index in Bytes to an Index in characters

Declaration: `function ByteToCharIndex(const S: String; Index: Integer) : Integer`

Visibility: `default`

Description: `ByteToCharIndex` returns the index (in characters) of the `Index`-th byte in `S`.

Errors: This function does not take into account MBCS yet.

See also: `CharToByteLen` ([1106](#)), `ByteToCharLen` ([1105](#))

ByteToCharLen

Synopsis: Convert a length in bytes to a length in characters.

Declaration: `function ByteToCharLen(const S: String;MaxLen: Integer) : Integer`

Visibility: default

Description: `ByteToCharLen` returns the number of bytes in `S`, but limits the result to `MaxLen`

Errors: This function does not take into account MBCS yet.

See also: `CharToByteLen` ([1106](#)), `ByteToCharIndex` ([1104](#))

ByteType

Synopsis: Return the type of byte in an ansistring for a multi-byte character set

Declaration: `function ByteType(const S: String;Index: Integer) : TMbcsByteType`

Visibility: default

Description: `ByteType` returns the type of byte in the ansistring `S` at (1-based) position `Index`.

Errors: No checking on the index is performed.

See also: `TMbcsByteType` ([1085](#)), `StrByteType` ([1168](#))

CallTerminateProcs

Synopsis: Call the exit chain procedures.

Declaration: `function CallTerminateProcs : Boolean`

Visibility: default

Description: `CallTerminateProcs` is run on program exit. It executes all terminate procedures that were added to the exit chain with `AddTerminateProc` ([1089](#)), and does this in reverse order.

Errors: If one of the exit procedure raises an exception, it is *not* caught, and the remaining exit procedures will not be executed.

See also: `TTerminateProc` ([1086](#)), `AddTerminateProc` ([1089](#))

ChangeFileExt

Synopsis: Change the extension of a filename.

Declaration: `function ChangeFileExt(const FileName: String;const Extension: String)
: String`

Visibility: default

Description: `ChangeFileExt` changes the file extension in `FileName` to `Extension`. The extension `Extension` includes the starting `.` (dot). The previous extension of `FileName` are all characters after the last `.`, the `.` character included.

If `FileName` doesn't have an extension, `Extension` is just appended.

Errors: None.

See also: `ExtractFileName` ([1124](#)), `ExtractFilePath` ([1124](#)), `ExpandFileName` ([1122](#))

CharToByteLen

Synopsis: Convert a length in characters to a length in bytes.

Declaration: `function CharToByteLen(const S: String;MaxLen: Integer) : Integer`

Visibility: default

Description: `CharToByteLen` returns the number of bytes in `S`, but limits the result to `MaxLen`

Errors: This function does not take into account MBCS yet.

See also: `ByteToCharLen` ([1105](#)), `ByteToCharIndex` ([1104](#))

CompareMem

Synopsis: Compare two memory areas.

Declaration: `function CompareMem(P1: Pointer;P2: Pointer;Length: cardinal) : Boolean`

Visibility: default

Description: `CompareMem` compares, byte by byte, 2 memory areas pointed to by `P1` and `P2`, for a length of `L` bytes.

It returns the following values:

<0 if at some position the byte at `P1` is less than the byte at the same position at `P2`.

0 if all `L` bytes are the same.

>0 if at some position the byte at `P1` is greater than the byte at the same position at `P2`.

Errors:

CompareMemRange

Synopsis: Compare 2 memory locations

Declaration: `function CompareMemRange(P1: Pointer;P2: Pointer;Length: cardinal)
: Integer`

Visibility: default

Description: CompareMemRange compares the 2 memory locations pointed to by P1 and P2 byte per byte. It stops comparing after Length bytes have been compared, or when it has encountered 2 different bytes. The result is then

>0 if a byte in range P1 was found that is bigger than the corresponding byte in range P2.

0 if all bytes in range P1 are the same as the corresponding bytes in range P2.

<0 if a byte in range P1 was found that is less than the corresponding byte in range P2.

Errors: None.

See also: SameText ([1165](#))

CompareStr

Synopsis: Compare 2 ansistrings case-sensitively, ignoring special characters.

Declaration: `function CompareStr(const S1: String;const S2: String) : Integer`

Visibility: default

Description: CompareStr compares two strings, S1 and S2, and returns the following result:

<0 if S1<S2.

0 if S1=S2.

>0 if S1>S2.

The comparison of the two strings is case-sensitive. The function does not take internationalization settings into account, it simply compares ASCII values.

Errors: None.

See also: AnsiCompareText ([1091](#)), AnsiCompareStr ([1090](#)), CompareText ([1108](#))

Listing: ./sysutex/ex65.pp

Program Example65;

```
{ This program demonstrates the CompareStr function }
{$H+}
```

Uses sysutils;

Procedure TestIt (S1,S2 : **String**);

Var R : Longint;

begin

 R:=CompareStr(S1,S2);

Write (' ',S1, ' is ');

```

If R<0 then
  write ( 'less than ' )
else If R=0 then
  Write ( 'equal to ' )
else
  Write ( 'larger than ' );
Writeln ( ' ', S2, ' ' );
end;

Begin
  Testit( 'One string ', 'One smaller string ' );
  Testit( 'One string ', 'one string ' );
  Testit( 'One string ', 'One string ' );
  Testit( 'One string ', 'One tall string ' );
End.

```

CompareText

Synopsis: Compare 2 ansistrings case insensitive.

Declaration: `function CompareText(const S1: String;const S2: String) : Integer`

Visibility: default

Description: CompareText compares two strings, S1 and S2, and returns the following result:

```

<0if S1<S2.
0if S1=S2.
>0if S1>S2.

```

The comparison of the two strings is case-insensitive. The function does not take internationalization settings into account, it simply compares ASCII values.

Errors: None.

See also: `AnsiCompareText` ([1091](#)), `AnsiCompareStr` ([1090](#)), `CompareStr` ([1107](#))

Listing: ./sysutex/ex66.pp

Program Example66;

```

{ This program demonstrates the CompareText function }
{$H+}

```

Uses sysutils;

Procedure TestIt (S1,S2 : **String**);

Var R : Longint;

```

begin
  R:=CompareText(S1,S2);
  Write ( ' ', S1, ' is ' );
  If R<0 then
    write ( 'less than ' )
  else If R=0 then
    Write ( 'equal to ' )

```

```

    else
        Write ( 'larger than ');
        WriteLn ( '', S2, '' );
    end;

Begin
    Testit( 'One string ', 'One smaller string ');
    Testit( 'One string ', 'one string ');
    Testit( 'One string ', 'One string ');
    Testit( 'One string ', 'One tall string ');
End.

```

CreateDir

Synopsis: Create a new directory

Declaration: `function CreateDir(const NewDir: String) : Boolean`

Visibility: default

Description: `CreateDir` creates a new directory with name `NewDir`. If the directory doesn't contain an absolute path, then the directory is created below the current working directory.

The function returns `True` if the directory was successfully created, `False` otherwise.

Errors: In case of an error, the function returns `False`.

See also: `RemoveDir` ([1163](#))

Listing: `./sysutex/ex26.pp`

Program `Example26`;

```

{ This program demonstrates the CreateDir and RemoveDir functions }
{ Run this program twice in the same directory }

```

Uses `sysutils`;

```

Begin
    If Not DirectoryExists( 'NewDir' ) then
        If Not CreateDir ( 'NewDir' ) Then
            WriteLn ( 'Failed to create directory !' )
        else
            WriteLn ( 'Created "NewDir" directory ' )
        Else
            If Not RemoveDir ( 'NewDir' ) Then
                WriteLn ( 'Failed to remove directory !' )
            else
                WriteLn ( 'Removed "NewDir" directory ' );
    End.

```

CurrToStr

Synopsis: Convert a currency value to a string.

Declaration: `function CurrToStr(Value: Currency) : String`

Visibility: default

Description: CurrToStr will convert a currency value to a string with a maximum of 15 digits, and precision 2. Calling CurrToStr is equivalent to calling FloatToStrF (1137):

```
FloatToStrF(Value, ffNumber, 15, 2);
```

Errors: None.

See also: FloatToStrF (1137), StrToCurr (1180)

Date

Synopsis: Return the current date.

Declaration: function Date : TDateTime

Visibility: default

Description: Date returns the current date in TDateTime format.

Errors: None.

See also: Time (1188), Now (1162)

Listing: ./sysutex/ex1.pp

Program Example1;

{ This program demonstrates the Date function }

uses sysutils;

Var YY,MM,DD : Word;

Begin

WriteIn ('Date : ',Date);

DeCodeDate (Date,YY,MM,DD);

WriteIn (**format** ('Date is (DD/MM/YY): %d/%d/%d ',[dd,mm,yy]));

End.

DateTimeToFileDate

Synopsis: Convert a TDateTime value to a file age (integer)

Declaration: function DateTimeToFileDate(DateTime: TDateTime) : LongInt

Visibility: default

Description: DateTimeToFileDate function converts a date/time indication in TDateTime format to a file-date function, such as returned for instance by the FileAge (1125) function.

Errors: None.

See also: Time (1188), Date (1110), FileDateToDateTime (1127), DateTimeToSystemTime (1112), DateTimeToTimeStamp (1113)

Listing: ./sysutex/ex2.pp

Program Example2;

{ This program demonstrates the DateTimeToFileDate function }

Uses sysutils;

Begin

WriteIn ('FileTime of now would be: ', **DateTimeToFileDate** (**Now**));
End.

DateTimeToStr

Synopsis: Converts a TDateTime value to a string using a predefined format.

Declaration: function DateTimeToStr(DateTime: TDateTime) : String

Visibility: default

Description: DateTimeToStr returns a string representation of DateTime using the formatting specified in ShortDateTimeFormat. It corresponds to a call to FormatDateTime('c',DateTime) (see formatchars (1074)).

Errors: None.

See also: FormatDateTime (1147)

Listing: ./sysutex/ex3.pp

Program Example3;

{ This program demonstrates the DateTimeToStr function }

Uses sysutils;

Begin

WriteIn ('Today is : ', **DateTimeToStr** (**Now**));
 WriteIn ('Today is : ', **FormatDateTime** ('c',**Now**));
End.

DateTimeToString

Synopsis: Converts a TDateTime value to a string with a given format.

Declaration: procedure DateTimeToString(var Result: String; const FormatStr: String;
const DateTime: TDateTime)

Visibility: default

Description: DateTimeToString returns in Result a string representation of DateTime using the formatting specified in FormatStr. for a list of characters that can be used in the FormatStr formatting string, see formatchars (1074).

Errors: In case a wrong formatting character is found, an EConvertError is raised.

See also: FormatDateTime (1147), formatchars (1074)

```

Uses sysutils;

Var ST : TSystemTime;

Begin
  DateTimeToSystemTime(Now,ST);
  With St do
    begin
      Writeln ( 'Today is      ',year,'/',month,'/',Day);
      Writeln ( 'The time is  ',Hour,':',minute,':',Second,'.',MilliSecond);
    end;
End.

```

DateTimeToTimeStamp

Synopsis: Converts a TDateTime value to a TimeStamp structure.

Declaration: function DateTimeToTimeStamp(DateTime: TDateTime) : TTimeStamp

Visibility: default

Description: DateTimeToSystemTime converts a date/time pair in DateTime, with TDateTime format to a TTimeStamp format.

Errors: None.

See also: DateTimeToFileDate ([1110](#)), SystemTimeToDateTime ([1186](#)), DateTimeToSystemTime ([1112](#))

Listing: ./sysutex/ex6.pp

Program Example6;

{ This program demonstrates the DateTimeToTimeStamp function }

Uses sysutils;

Var TS : TTimeStamp;

Begin

TS:=DateTimeToTimeStamp (**Now**);

With TS **do**

begin

Writeln ('Now is ',**time**, ' millisecond past midnight');

Writeln ('Today is ',**Date**, ' days past 1/1/0001');

end;

End.

DateToStr

Synopsis: Converts a TDateTime value to a date string with a predefined format.

Declaration: function DateToStr(Date: TDateTime) : String

Visibility: default

Description: DateToStr converts Date to a string representation. It uses ShortDateFormat as it's formatting string. It is hence completely equivalent to a FormatDateTime('dddd', Date).

Errors: None.

See also: [TimeToStr \(1189\)](#), [DateTimeToStr \(1111\)](#), [FormatDateTime \(1147\)](#), [StrToDate \(1180\)](#)

Listing: ./sysutex/ex7.pp

Program Example7;

{ This program demonstrates the DateToStr function }

Uses sysutils;

Begin

WriteIn (**Format** ('Today is: %s' ,[**DateToStr** (**Date**)]));

End.

DayOfWeek

Synopsis: Returns the day of the week.

Declaration: `function DayOfWeek(DateTime: TDateTime) : Integer`

Visibility: default

Description: `DayOfWeek` returns the day of the week from `DateTime`. Sunday is counted as day 1, Saturday is counted as day 7. The result of `DayOfWeek` can serve as an index to the `LongDayNames` constant array, to retrieve the name of the day.

Errors: None.

See also: [Date \(1110\)](#), [DateToStr \(1113\)](#)

Listing: ./sysutex/ex8.pp

Program Example8;

{ This program demonstrates the DayOfWeek function }

Uses sysutils;

Begin

WriteIn ('Today 's day is ' ,LongDayNames[**DayOfWeek** (**Date**)]);

End.

DecodeDate

Synopsis: Decode a `TDateTime` to a year,month,day triplet

Declaration: `procedure DecodeDate(Date: TDateTime;var Year: Word;var Month: Word;var Day: Word)`

Visibility: default

Description: `DecodeDate` decodes the Year, Month and Day stored in `Date`, and returns them in the Year, Month and Day variables.

Errors: None.

See also: [EncodeDate \(1119\)](#), [DecodeTime \(1115\)](#)

Listing: ./sysutex/ex9.pp

```

Program Example9;

{ This program demonstrates the DecodeDate function }

Uses sysutils;

Var YY,MM,DD : Word;

Begin
  DecodeDate( Date ,YY,MM,DD);
  WriteLn ( Format ( 'Today is %d/%d/%d' ,[dd,mm,yy] ));
End.

```

DecodeDateFully

Synopsis: Decode a date with additional date of the week.

Declaration: `function DecodeDateFully(const DateTime: TDateTime;var Year: Word;
var Month: Word;var Day: Word;var DOW: Word)
: Boolean`

Visibility: default

Description: DecodeDateFully, like DecodeDate ([1114](#)), decodes DateTime in its parts and returns these in Year, Month, Day but in addition returns the day of the week in DOW.

Errors: None.

See also: [EncodeDate \(1119\)](#), [TryEncodeDate \(1191\)](#), [DecodeDate \(1114\)](#)

DecodeTime

Synopsis: Decode a TDateTime to a hour,minute,second,millisecond quartet

Declaration: `procedure DecodeTime(Time: TDateTime;var Hour: Word;var Minute: Word;
var Second: Word;var MilliSecond: Word)`

Visibility: default

Description: DecodeDate decodes the hours, minutes, second and milliseconds stored in Time, and returns them in the Hour, Minute and Second and MilliSecond variables.

Errors: None.

See also: [EncodeTime \(1119\)](#), [DecodeDate \(1114\)](#)

Listing: ./sysutex/ex10.pp

```

Program Example10;

{ This program demonstrates the DecodeTime function }

Uses sysutils;

```



```
Var HH,MM,SS,MS: Word;
```

```
Begin
```

```
  DecodeTime(Time,HH,MM,SS,MS);
```

```
  WriteLn (format('The time is %d:%d:%d.%d',[hh,mm,ss,ms]));
```

```
End.
```

DeleteFile

Synopsis: Delete a file from the filesystem.

Declaration: `function DeleteFile(const FileName: String) : Boolean`

Visibility: default

Description: `DeleteFile` deletes file `FileName` from disk. The function returns `True` if the file was successfully removed, `False` otherwise.

Errors: On error, `False` is returned.

See also: `FileCreate` ([1126](#)), `FileExists` ([1128](#))

Listing: `./sysutex/ex31.pp`

Program `Example31`;

```
{ This program demonstrates the DeleteFile function }
```

```
Uses sysutils;
```

```
Var
```

```
  Line : String;
```

```
  F,I : Longint;
```

```
Begin
```

```
  F:=FileCreate('test.txt');
```

```
  Line:='Some string line.'#10;
```

```
  For I:=1 to 10 do
```

```
    FileWrite (F,Line[I],Length(Line));
```

```
  FileClose(F);
```

```
  DeleteFile('test.txt');
```

```
End.
```

DirectoryExists

Synopsis: Check whether a directory exists in the file system.

Declaration: `function DirectoryExists(const Directory: String) : Boolean`

Visibility: default

Description: `DirectoryExists` checks whether `Directory` exists in the filesystem and is actually a directory. If this is the case, the function returns `True`, otherwise `False` is returned.

See also: `FileExists` ([1128](#))

DiskFree

Synopsis: Return the amount of free disk space

Declaration: `function DiskFree(drive: Byte) : Int64`

Visibility: default

Description: `DiskFree` returns the free space (in bytes) on disk `Drive`. `Drive` is the number of the disk drive:

- 0** for the current drive.
- 1** for the first floppy drive.
- 2** for the second floppy drive.
- 3** for the first hard-disk partition.
- 4-26** for all other drives and partitions.

Remark: Under linux, and Unix in general, the concept of disk is different than the dos one, since the filesystem is seen as one big directory tree. For this reason, the `DiskFree` and `DiskSize` (1117) functions must be mimicked using filenames that reside on the partitions. For more information, see `AddDisk` (1088).

Errors: On error, -1 is returned.

See also: `DiskSize` (1117), `AddDisk` (1088)

Listing: `./sysutex/ex27.pp`

Program `Example27`;

{ This program demonstrates the DiskFree function }

Uses `sysutils`;

Begin

```
Write ('Size of current disk      : ', DiskSize(0));
WriteLn (' (= ', DiskSize(0) div 1024, 'k) ');
Write ('Free space of current disk : ', DiskFree(0));
WriteLn (' (= ', DiskFree(0) div 1024, 'k) ');
```

End.

DiskSize

Synopsis: Return the total amount of disk space.

Declaration: `function DiskSize(drive: Byte) : Int64`

Visibility: default

Description: `DiskSize` returns the size (in bytes) of disk `Drive`. `Drive` is the number of the disk drive:

- 0** for the current drive.
- 1** for the first floppy drive.
- 2** for the second floppy drive.
- 3** for the first hard-disk partition.
- 4-26** for all other drives and partitions.

Remark: Under linux, and Unix in general, the concept of disk is different than the dos one, since the filesystem is seen as one big directory tree. For this reason, the `DiskFree` (1117) and `DiskSize` functions must be mimicked using filenames that reside on the partitions. For more information, see `AddDisk` (1088)

For an example, see `DiskFree` (1117).

Errors: On error, -1 is returned.

See also: `DiskFree` (1117), `AddDisk` (1088)

DisposeStr

Synopsis: Dispose an ansistring from the heap.

Declaration: `procedure DisposeStr(S: PString)`

Visibility: default

Description: `DisposeStr` removes the dynamically allocated string `S` from the heap, and releases the occupied memory.

This function is provided for Delphi compatibility only. `AnsiStrings` are managed on the heap and should be preferred to the mechanism of dynamically allocated strings.

For an example, see `DisposeStr` (1118).

Errors: None.

See also: `NewStr` (1161), `AppendStr` (1102), `AssignStr` (1103)

DoDirSeparators

Synopsis: Convert known directory separators to the current directory separator.

Declaration: `procedure DoDirSeparators(var FileName: String)`

Visibility: default

Description: This function replaces all known directory separators to the directory separator character for the current system.

Errors: None.

See also: `ExtractFileName` (1124), `ExtractFilePath` (1124)

Listing: `./sysutex/ex32.pp`

Program Example32;

```
{ This program demonstrates the DoDirSeparators function }
{$H+}
```

Uses sysutils;

Procedure Testit (F : **String**);

begin

```
  WriteLn ( 'Before : ',F);
  DoDirSeparators (F);
```

```

    WriteLn ( 'After   : ',F);
end;

Begin
    Testit ( GetCurrentDir );
    Testit ( 'c:\pp\bin\win32' );
    Testit ( '/usr/lib/fpc' );
    Testit ( '\usr\lib\fpc' );
End.

```

EncodeDate

Synopsis: Encode a Year,Month,Day to a TDateTime value.

Declaration: `function EncodeDate(Year: Word;Month: Word;Day: Word) : TDateTime`

Visibility: default

Description: EncodeDate encodes the Year, Month and Day variables to a date in TDateTime format. It does the opposite of the DecodeDate (1114) procedure.

The parameters must lie withing valid ranges (boundaries included):

Year must be between 1 and 9999.

Month must be within the range 1-12.

Days must be between 1 and 31.

Errors: In case one of the parameters is out of it's valid range, 0 is returned.

See also: EncodeTime (1119), DecodeDate (1114)

Listing: ./sysutex/ex11.pp

Program Example11;

{ This program demonstrates the EncodeDate function }

Uses sysutils;

Var YY,MM,DD : Word;

Begin

DecodeDate (Date,YY,MM,DD);

WriteLn ('Today is : ',**FormatDateTime** ('dd mmm yyyy',**EnCodeDate**(YY,Mm,Dd)));

End.

EncodeTime

Synopsis: Encode a Hour,Min,Sec,millisecond to a TDateTime value.

Declaration: `function EncodeTime(Hour: Word;Minute: Word;Second: Word;MilliSecond: Word) : TDateTime`

Visibility: default

Description: EncodeTime encodes the Hour, Minute, Second, MilliSecond variables to a TDateTime format result. It does the opposite of the DecodeTime (1115) procedure.

The parameters must have a valid range (boundaries included):

Hour must be between 0 and 23.

Minute, second must both be between 0 and 59.

Millisecond must be between 0 and 999.

Errors: In case one of the parameters is outside of it's valid range, 0 is returned.

See also: EncodeDate (1119), DecodeTime (1115)

Listing: ./sysutex/ex12.pp

Program Example12;

{ This program demonstrates the EncodeTime function }

Uses sysutils;

Var Hh,MM,SS,MS : Word;

Begin

DeCodeTime (Time, Hh, MM, SS, MS);

WriteIn ('Present Time is : ', **FormatDateTime** ('hh:mm:ss ', **EnCodeTime** (HH, MM, SS, MS)));

End.

ExceptAddr

Synopsis: Current exception address.

Declaration: function ExceptAddr : Pointer

Visibility: default

Description: ExceptAddr returns the address from the currently treated exception object when an exception is raised, and the stack is unwound.

See also: ExceptObject (1121), ExceptionErrorMessage (1120), ShowException (1166)

ExceptionErrorMessage

Synopsis: Return a message describing the exception.

Declaration: function ExceptionErrorMessage(ExceptObject: TObject;
ExceptAddr: Pointer; Buffer: PChar;
Size: Integer) : Integer

Visibility: default

Description: ExceptionErrorMessage creates a string that describes the exception object ExceptObject at address ExceptAddr. It can be used to display exception messages. The string will be stored in the memory pointed to by Buffer, and will at most have Size characters.

The routine checks whether ExceptObject is a Exception (1199) object or not, and adapts the output accordingly.

See also: ExceptObject (1121), ExceptAddr (1120), ShowException (1166)

ExceptObject

Synopsis: Current Exception object.

Declaration: `function ExceptObject : TObject`

Visibility: default

Description: `ExceptObject` returns the currently treated exception object when an exception is raised, and the stack is unwound.

Errors: If there is no exception, the function returns `Nil`

See also: `ExceptAddr` (1120), `ExceptionErrorMessage` (1120), `ShowException` (1166)

ExcludeTrailingBackslash

Synopsis: Strip trailing directory separator from a pathname, if needed.

Declaration: `function ExcludeTrailingBackslash(const Path: String) : String`

Visibility: default

Description: `ExcludeTrailingBackslash` is provided for backwards compatibility with Delphi. Use `ExcludeTrailingPathDelimiter` (1121) instead.

See also: `IncludeTrailingPathDelimiter` (1154), `ExcludeTrailingPathDelimiter` (1121), `PathDelim` (1081), `IsPathDelimiter` (1158)

ExcludeTrailingPathDelimiter

Synopsis: Strip trailing directory separator from a pathname, if needed.

Declaration: `function ExcludeTrailingPathDelimiter(const Path: String) : String`

Visibility: default

Description: `ExcludeTrailingPathDelimiter` removes the trailing path delimiter character (`PathDelim` (1081)) from `Path` if it is present, and returns the result.

See also: `ExcludeTrailingBackslash` (1121), `IncludeTrailingPathDelimiter` (1154), `PathDelim` (1081), `IsPathDelimiter` (1158)

ExecuteProcess

Synopsis: Execute another process (program).

Declaration: `function ExecuteProcess(const Path: AnsiString;
 const ComLine: AnsiString) : Integer
function ExecuteProcess(const Path: AnsiString;
 const ComLine: Array[] of AnsiString) : Integer`

Visibility: default

Description: `ExecuteProcess` will execute the program in `Path`, passing it the arguments in `ComLine`. `ExecuteProcess` will then wait for the program to finish, and will return the exit code of the executed program. In case `ComLine` is a single string, it will be split out in an array of strings, taking into account common whitespace and quote rules.

Errors: In case the program could not be executed or an other error occurs, an `EOSError` (1197) exception will be raised.

See also: `EOSError` (1197)

ExpandFileName

Synopsis: Expand a relative filename to an absolute filename.

Declaration: `function ExpandFileName(const FileName: String) : String`

Visibility: default

Description: `ExpandFileName` expands the filename to an absolute filename. It changes all directory separator characters to the one appropriate for the system first.

Errors: None.

See also: `ExtractFileName` (1124), `ExtractFilePath` (1124), `ExtractFileDir` (1123), `ExtractFileDrive` (1123), `ExtractFileExt` (1124), `ExtractRelativePath` (1125)

Listing: `./sysutex/ex33.pp`

Program Example33;

{ This program demonstrates the ExpandFileName function }

Uses sysutils;

Procedure Testit (F : **String**);

begin

WriteLn (F, ' expands to : ', **ExpandFileName**(F));

end;

Begin

 Testit('ex33.pp');
 Testit(**ParamStr**(0));
 Testit('/pp/bin/win32/ppc386');
 Testit('\pp\bin\win32\ppc386');
 Testit('.');

End.

ExpandUNCFileName

Synopsis: Expand a relative filename to an absolute UNC filename.

Declaration: `function ExpandUNCFileName(const FileName: String) : String`

Visibility: default

Description: `ExpandUNCFileName` runs `ExpandFileName` (1122) on `FileName` and then attempts to replace the driveletter by the name of a shared disk.

Errors: None.

See also: `ExtractFileName` (1124), `ExtractFilePath` (1124), `ExtractFileDir` (1123), `ExtractFileDrive` (1123), `ExtractFileExt` (1124), `ExtractRelativePath` (1125)

ExtractFileDir

Synopsis: Extract the directory part of a filename.

Declaration: `function ExtractFileDir(const FileName: String) : String`

Visibility: default

Description: `ExtractFileDir` returns only the directory part of `FileName`, not including a driveletter. The directory name has NO ending directory separator, in difference with `ExtractFilePath` (1124).

Errors: None.

See also: `ExtractFileName` (1124), `ExtractFilePath` (1124), `ExtractFileDir` (1123), `ExtractFileDrive` (1123), `ExtractFileExt` (1124), `ExtractRelativePath` (1125)

Listing: `./sysutex/ex34.pp`

Program `Example34`;

```
{ This program demonstrates the ExtractFileName function }
{$H+}
Uses sysutils;
```

```
Procedure Testit(F : String);
```

```
begin
  WriteLn ( 'FileName      : ', F);
  WriteLn ( 'Has Name      : ', ExtractFileName(F));
  WriteLn ( 'Has Path      : ', ExtractFilePath(F));
  WriteLn ( 'Has Extension : ', ExtractFileExt(F));
  WriteLn ( 'Has Directory : ', ExtractFileDir(F));
  WriteLn ( 'Has Drive     : ', ExtractFileDrive(F));
end;
```

```
Begin
  Testit ( Paramstr(0));
  Testit ( '/usr/local/bin/mysqld' );
  Testit ( 'c:\pp\bin\win32\ppc386.exe' );
  Testit ( '/pp/bin/win32/ppc386.exe' );
End.
```

ExtractFileDrive

Synopsis: Extract the drive part from a filename.

Declaration: `function ExtractFileDrive(const FileName: String) : String`

Visibility: default

Description: `Extracts` the drive letter from a filename. Note that some operating systems do not support drive letters.

For an example, see `ExtractFileDir` (1123).

Errors:

See also: `ExtractFileName` (1124), `ExtractFilePath` (1124), `ExtractFileDir` (1123), `ExtractFileDrive` (1123), `ExtractFileExt` (1124), `ExtractRelativePath` (1125)

ExtractFileExt

Synopsis: Return the extension from a filename.

Declaration: `function ExtractFileExt(const FileName: String) : String`

Visibility: default

Description: `ExtractFileExt` returns the extension (including the `.` (dot) character) of `FileName`.

For an example, see `ExtractFileDir` (1123).

Errors: None.

See also: `ExtractFileName` (1124), `ExtractFilePath` (1124), `ExtractFileDir` (1123), `ExtractFileDrive` (1123), `ExtractFileExt` (1124), `ExtractRelativePath` (1125)

ExtractFileName

Synopsis: Extract the filename part from a full path filename.

Declaration: `function ExtractFileName(const FileName: String) : String`

Visibility: default

Description: `ExtractFileName` returns the filename part from `FileName`. The filename consists of all characters after the last directory separator character (`'/'` or `'\'`) or drive letter.

The full filename can always be reconstructed by concatenating the result of `ExtractFilePath` (1124) and `ExtractFileName`.

For an example, see `ExtractFileDir` (1123).

Errors: None.

See also: `ExtractFileName` (1124), `ExtractFilePath` (1124), `ExtractFileDir` (1123), `ExtractFileDrive` (1123), `ExtractFileExt` (1124), `ExtractRelativePath` (1125)

ExtractFilePath

Synopsis: Extract the path from a filename.

Declaration: `function ExtractFilePath(const FileName: String) : String`

Visibility: default

Description: `ExtractFilePath` returns the path part (including driveletter) from `FileName`. The path consists of all characters before the last directory separator character (`'/'` or `'\'`), including the directory separator itself. In case there is only a drive letter, that will be returned.

The full filename can always be reconstructed by concatenating the result of `ExtractFilePath` and `ExtractFileName` (1124).

For an example, see `ExtractFileDir` (1123).

Errors: None.

See also: `ExtractFileName` (1124), `ExtractFilePath` (1124), `ExtractFileDir` (1123), `ExtractFileDrive` (1123), `ExtractFileExt` (1124), `ExtractRelativePath` (1125)

ExtractRelativepath

Synopsis: Extract a relative path from a filename, given a base directory.

Declaration: `function ExtractRelativepath(const BaseName: String;
const DestName: String) : String`

Visibility: default

Description: `ExtractRelativePath` constructs a relative path to go from `BaseName` to `DestName`. If `DestName` is on another drive (Not on Linux) then the whole `Destname` is returned. *Note:* This function does not exist in the Delphi unit.

Errors: None.

See also: `ExtractFileName` ([1124](#)), `ExtractFilePath` ([1124](#)), `ExtractFileDir` ([1123](#)), `ExtractFileDrive` ([1123](#)), `ExtractFileExt` ([1124](#))

Listing: ./sysutex/ex35.pp

Program Example35;

{ This program demonstrates the ExtractRelativePath function }

Uses sysutils;

Procedure Testit (FromDir, ToDir : **String**);

begin

Write ('From " ', FromDir, '" to " ', ToDir, '" via " ');

WriteLn (ExtractRelativePath (FromDir, ToDir), ' " ');

end;

Begin

 Testit ('/pp/src/compiler ', '/pp/bin/win32/ppc386');

 Testit ('/pp/bin/win32/ppc386 ', '/pp/src/compiler');

 Testit ('e:/pp/bin/win32/ppc386 ', 'd:/pp/src/compiler');

 Testit ('e:\pp\bin\win32\ppc386 ', 'd:\pp\src\compiler');

End.

FileAge

Synopsis: Return the timestamp of a file.

Declaration: `function FileAge(const FileName: String) : LongInt`

Visibility: default

Description: `FileAge` returns the last modification time of file `FileName`. The `FileDate` format can be transformed to `TDateTime` format with the `FileDateToDateTime` ([1127](#)) function.

Errors: In case of errors, -1 is returned.

See also: `FileDateToDateTime` ([1127](#)), `FileExists` ([1128](#)), `FileGetAttr` ([1128](#))

Listing: ./sysutex/ex36.pp

Program Example36;

{ This program demonstrates the FileAge function }

Uses sysutils;

Var S : TDateTime;

fa : Longint;

Begin

fa:=FileAge('ex36.pp');

If Fa<>-1 **then**

begin

S:=FileDateTodateTime(fa);

WriteLn ('I'm from ',DateTimeToStr(S))

end;

End.

FileClose

Synopsis: Close a file handle.

Declaration: procedure FileClose(Handle: LongInt)

Visibility: default

Description: FileClose closes the file handle Handle. After this call, attempting to read or write from the handle will result in an error.

For an example, see FileCreate ([1126](#))

Errors: None.

See also: FileCreate ([1126](#)), FileWrite ([1133](#)), FileOpen ([1130](#)), FileRead ([1131](#)), FileTruncate ([1133](#)), FileSeek ([1132](#))

FileCreate

Synopsis: Create a new file and return a handle to it.

Declaration: function FileCreate(const FileName: String) : LongInt

function FileCreate(const FileName: String;Mode: Integer) : LongInt

Visibility: default

Description: FileCreate creates a new file with name FileName on the disk and returns a file handle which can be used to read or write from the file with the FileRead ([1131](#)) and FileWrite ([1133](#)) functions. If a file with name FileName already existed on the disk, it is overwritten.

Errors: If an error occurs (e.g. disk full or non-existent path), the function returns -1.

See also: FileClose ([1126](#)), FileWrite ([1133](#)), FileOpen ([1130](#)), FileRead ([1131](#)), FileTruncate ([1133](#)), FileSeek ([1132](#))

Listing: ./sysutex/ex37.pp

```

Program Example37;

{ This program demonstrates the FileCreate function }

Uses sysutils;

Var I,J,F : Longint;

Begin
  F:=FileCreate ( 'test.dat' );
  If F=-1 then
    Halt (1);
  For I:=0 to 100 do
    FileWrite(F,I,SizeOf(i));
  FileClose(f);
  F:=FileOpen ( 'test.dat',fmOpenRead);
  For I:=0 to 100 do
    begin
      FileRead (F,J,SizeOf(J));
      If J<>I then
        Writeln ( 'Mismatch at file position ',I)
      end;
    FileSeek(F,0,fsFromBeginning);
    Randomize;
    Repeat
      FileSeek(F,Random(100)*4,fsFromBeginning);
      FileRead (F,J,SizeOf(J));
      Writeln ( 'Random read : ',j);
    Until J>80;
    FileClose(F);
    F:=FileOpen( 'test.dat',fmOpenWrite);
    I:=50*SizeOf(Longint);
    If FileTruncate(F,I) then
      Writeln('Successfully truncated file to ',I,' bytes. ');
    FileClose(F);
End.

```

FileDateToDateTime

Synopsis: Convert a FileDate value to a TDateTime value.

Declaration: function FileDateToDateTime(Filedate: LongInt) : TDateTime

Visibility: default

Description: FileDateToDateTime converts the date/time encoded in filedate to a TDateTime encoded form. It can be used to convert date/time values returned by the FileAge ([1125](#)) or FindFirst ([1134](#))/FindNext ([1135](#)) functions to TDateTime form.

Errors: None.

See also: DateTimeToFileDate ([1110](#))

Listing: ./sysutex/ex13.pp

Program Example13;

```

{ This program demonstrates the FileDateToDateTime function }

Uses sysutils;

Var
    ThisAge : Longint;

Begin
    Write ( 'ex13.pp created on : ');
    ThisAge:=FileAge( 'ex13.pp' );
    Writeln ( DateTimeToStr( FileDateToDateTime( ThisAge ) ));
End.

```

FileExists

Synopsis: Check whether a file exists in the filesystem.

Declaration: `function FileExists(const FileName: String) : Boolean`

Visibility: default

Description: `FileExists` returns `True` if a file with name `FileName` exists on the disk, `False` otherwise.

Errors: None.

See also: `FileAge` ([1125](#)), `FileGetAttr` ([1128](#)), `FileSetAttr` ([1133](#))

Listing: ./sysutex/ex38.pp

Program Example38;

```

{ This program demonstrates the FileExists function }

Uses sysutils;

Begin
    If FileExists(ParamStr(0)) Then
        Writeln ( 'All is well , I seem to exist.' );
End.

```

FileGetAttr

Synopsis: Return attributes of a file.

Declaration: `function FileGetAttr(const FileName: String) : LongInt`

Visibility: default

Description: `FileGetAttr` returns the attribute settings of file `FileName`. The attribute is a OR-ed combination of the following constants:

faReadOnly The file is read-only.

faHidden The file is hidden. (On unix, this means that the filename starts with a dot)

faSysFile The file is a system file (On unix, this means that the file is a character, block or FIFO file).

faVolumeId Volume Label. Not possible under unix.

faDirectoryFile is a directory.

faArchivefile is an archive. Not possible on Unix

Errors: In case of error, -1 is returned.

See also: FileSetAttr ([1133](#)), FileAge ([1125](#)), FileGetDate ([1129](#))

Listing: ./sysutex/ex40.pp

Program Example40;

{ This program demonstrates the FileGetAttr function }

Uses sysutils;

Procedure Testit (**Name** : **String**);

Var F : Longint;

Begin

 F:= FileGetAttr (**Name**);

If F<>-1 **then**

begin

 WriteLn ('Testing : ',**Name**);

If (F **and** faReadOnly)<>0 **then**

 WriteLn ('File is ReadOnly');

If (F **and** faHidden)<>0 **then**

 WriteLn ('File is hidden');

If (F **and** faSysFile)<>0 **then**

 WriteLn ('File is a system file');

If (F **and** faVolumeID)<>0 **then**

 WriteLn ('File is a disk label');

If (F **and** faArchive)<>0 **then**

 WriteLn ('File is artchive file');

If (F **and** faDirectory)<>0 **then**

 WriteLn ('File is a directory');

end

else

 WriteLn ('Error reading attribites of ',**Name**);

end;

begin

 testit ('ex40.pp');

 testit (**ParamStr**(0));

 testit ('. ');

 testit ('/ ');

End.

FileGetDate

Synopsis: Return the file time of an opened file.

Declaration: function FileGetDate(Handle: LongInt) : LongInt

Visibility: default

Description: `FileGetdate` returns the filetime of the opened file with filehandle `Handle`. It is the same as `FileAge` (1125), with this difference that `FileAge` only needs the file name, while `FileGetDate` needs an open file handle.

Errors: On error, -1 is returned.

See also: `FileAge` (1125)

Listing: `./sysutex/ex39.pp`

Program `Example39`;

{ This program demonstrates the FileGetDate function }

Uses `sysutils`;

Var `F,D` : `Longint`;

Begin

`F:=FileCreate('test.dat');`

`D:=FileGetDate(F);`

`WriteLn('File created on ',DateTimeToStr(FileDateToDateTime(D)));`

`FileClose(F);`

`DeleteFile('test.dat');`

End.

FileIsReadOnly

Synopsis: Check whether a file is read-only.

Declaration: `function FileIsReadOnly(const FileName: String) : Boolean`

Visibility: `default`

Description: `FileIsReadOnly` checks whether `FileName` exists in the filesystem and is a read-only file. If this is the case, the function returns `True`, otherwise `False` is returned.

See also: `FileExists` (1128)

FileOpen

Synopsis: Open an existing file and return a filehandle

Declaration: `function FileOpen(const FileName: String;Mode: Integer) : LongInt`

Visibility: `default`

Description: `FileOpen` opens a file with name `FileName` with mode `Mode`. `Mode` can be one of the following constants:

fmOpenReadThe file is opened for reading.

fmOpenWriteThe file is opened for writing.

fmOpenReadWriteThe file is opened for reading and writing.

If the file has been successfully opened, it can be read from or written to (depending on the Mode parameter) with the `FileRead` (1131) and `FileWrite` functions.

Remark: Remark that you cannot open a file if it doesn't exist yet, i.e. it will not be created for you. If you want to create a new file, or overwrite an old one, use the `FileCreate` (1126) function.

For an example, see `FileOpen` (1130)

Errors: On Error, -1 is returned.

See also: `FileClose` (1126), `FileWrite` (1133), `FileCreate` (1126), `FileRead` (1131), `FileTruncate` (1133), `FileSeek` (1132)

FileRead

Synopsis: Read data from a filehandle in a buffer.

Declaration: `function FileRead(Handle: LongInt;var Buffer;Count: LongInt) : LongInt`

Visibility: default

Description: `FileRead` reads Count bytes from file-handle Handle and stores them into Buffer. Buffer must be at least Count bytes long. No checking on this is performed, so be careful not to overwrite any memory. Handle must be the result of a `FileOpen` (1130) call.

For an example, see `FileCreate` (1126)

Errors: On error, -1 is returned.

See also: `FileClose` (1126), `FileWrite` (1133), `FileCreate` (1126), `FileOpen` (1130), `FileTruncate` (1133), `FileSeek` (1132)

FileSearch

Synopsis: Search for a file in a path.

Declaration: `function FileSearch(const Name: String;const DirList: String) : String`

Visibility: default

Description: `FileSearch` looks for the file Name in DirList, where dirlist is a list of directories, separated by semicolons or colons. It returns the full filename of the first match found.

Errors: On error, an empty string is returned.

See also: `ExpandFileName` (1122), `FindFirst` (1134)

Listing: ./sysutex/ex41.pp

Program Example41;

{ Program to demonstrate the FileSearch function. }

Uses Sysutils;

Const

```
{ $ifdef unix }
  FN = 'find';
  P = ' ./ bin :/ usr / bin ';
{ $else }
```

```

    FN = 'find.exe';
    P = 'c:\dos;c:\windows;c:\windows\system;c:\windows\system32';
{$endif}

begin
    Writeln ('find is in : ',FileSearch (FN,P));
end.

```

FileSeek

Synopsis: Set the current file position on a file handle.

Declaration: `function FileSeek(Handle: LongInt; FOffset: LongInt; Origin: LongInt) : LongInt`
`function FileSeek(Handle: LongInt; FOffset: Int64; Origin: Int64) : Int64`

Visibility: default

Description: FileSeek sets the file pointer on position `Offset`, starting from `Origin`. `Origin` can be one of the following values:

fsFromBeginning `Offset` is relative to the first byte of the file. This position is zero-based. i.e. the first byte is at offset 0.

fsFromCurrent `Offset` is relative to the current position.

fsFromEnd `Offset` is relative to the end of the file. This means that `Offset` can only be zero or negative in this case.

If successful, the function returns the new file position, relative to the beginning of the file.

Remark: The abovementioned constants do not exist in Delphi.

Errors: On error, -1 is returned.

See also: FileClose ([1126](#)), FileWrite ([1133](#)), FileCreate ([1126](#)), FileOpen ([1130](#)), FileRead ([1131](#)), FileTruncate ([1133](#))

Listing: ./sysutex/ex42.pp

Program Example42;

{ This program demonstrates the FileSetAttr function }

Uses sysutils;

Begin

```

    If FileSetAttr ('ex40.pp',faReadOnly or faHidden)=0 then
        Writeln ('Successfully made file hidden and read-only.')
    else
        Writeln ('Couldn't make file hidden and read-only.');
```

End.

FileSetAttr

Synopsis: Set the attributes of a file.

Declaration: `function FileSetAttr(const Filename: String;Attr: LongInt) : LongInt`

Visibility: default

Description: `FileSetAttr` sets the attributes of `FileName` to `Attr`. If the function was successful, 0 is returned, -1 otherwise. `Attr` can be set to an OR-ed combination of the pre-defined `faXXX` constants.

This function is not implemented on Unixes.

Errors: On error, -1 is returned (always on Unixes).

See also: `FileGetAttr` ([1128](#)), `FileGetDate` ([1129](#)), `FileSetDate` ([1133](#))

FileSetDate

Synopsis: Set the date of a file.

Declaration: `function FileSetDate(Handle: LongInt;Age: LongInt) : LongInt`

Visibility: default

Description: `FileSetDate` sets the file date of the file with handle `Handle` to `Age`, where `Age` is a DOS date-and-time stamp value.

The function returns zero if successfull. (not on unixes, where it is not implemented)

Errors: On Unix, -1 is always returned, since this is impossible to implement. On Windows and DOS, a negative error code is returned.

FileTruncate

Synopsis: Truncate an open file to a given size.

Declaration: `function FileTruncate(Handle: LongInt;Size: LongInt) : Boolean`

Visibility: default

Description: `FileTruncate` truncates the file with handle `Handle` to `Size` bytes. The file must have been opened for writing prior to this call. The function returns `True` is successful, `False` otherwise.

For an example, see `FileCreate` ([1126](#)).

Errors: On error, the function returns `False`.

See also: `FileClose` ([1126](#)), `FileWrite` ([1133](#)), `FileCreate` ([1126](#)), `FileOpen` ([1130](#)), `FileRead` ([1131](#)), `FileSeek` ([1132](#))

FileWrite

Synopsis: Write data from a buffer to a given filehandle.

Declaration: `function FileWrite(Handle: LongInt;const Buffer;Count: LongInt)
: LongInt`

Visibility: default

Description: `FileWrite` writes `Count` bytes from `Buffer` to the file with handle `Handle`. Prior to this call, the file must have been opened for writing. `Buffer` must be at least `Count` bytes large, or a memory access error may occur.

The function returns the number of bytes written, or -1 in case of an error.

For an example, see `FileCreate` (1126).

Errors: In case of error, -1 is returned.

See also: `FileClose` (1126), `FileCreate` (1126), `FileOpen` (1130), `FileRead` (1131), `FileTruncate` (1133), `FileSeek` (1132)

FindClose

Synopsis: Close a find handle

Declaration: `procedure FindClose(var F: TSearchRec)`

Visibility: default

Description: `FindClose` ends a series of `FindFirst` (1134)/`FindNext` (1135) calls, and frees any memory used by these calls. It is *absolutely* necessary to do this call, or huge memory losses may occur.

For an example, see `FindFirst` (1134).

Errors: None.

See also: `FindFirst` (1134), `FindNext` (1135)

FindCmdLineSwitch

Synopsis: Check whether a certain switch is present on the command-line.

Declaration: `function FindCmdLineSwitch(const Switch: String;
 const Chars: TSysCharSet; IgnoreCase: Boolean)
 : Boolean
 function FindCmdLineSwitch(const Switch: String; IgnoreCase: Boolean)
 : Boolean
 function FindCmdLineSwitch(const Switch: String) : Boolean`

Visibility: default

Description: `FindCmdLineSwitch` will check all command-line arguments for the presence of the option `Switch`. It will return `True` if it was found, `False` otherwise. Characters that appear in `Chars` (default is `SwitchChars` (1081)) are assumed to indicate an option (switch). If the parameter `IgnoreCase` is `True`, case will be ignored when looking for the switch. Default is to search case sensitive.

Errors: None.

See also: `SwitchChars` (1081)

FindFirst

Synopsis: Start a file search and return a findhandle

Declaration: `function FindFirst(const Path: String; Attr: LongInt;
 var Rslt: TSearchRec) : LongInt`

Visibility: default

Description: `FindFirst` looks for files that match the name (possibly with wildcards) in `Path` and attributes `Attr`. It then fills up the `Rslt` record with data gathered about the file. It returns 0 if a file matching the specified criteria is found, a nonzero value (-1 on linux) otherwise.

The `Rslt` record can be fed to subsequent calls to `FindNext`, in order to find other files matching the specifications.

Remark: A `FindFirst` call must *always* be followed by a `FindClose` (1134) call with the same `Rslt` record. Failure to do so will result in memory loss.

Errors: On error the function returns -1 on linux, a nonzero error code on Windows.

See also: `FindClose` (1134), `FindNext` (1135)

Listing: ./sysutex/ex43.pp

Program Example43;

{ This program demonstrates the FindFirst function }

Uses SysUtils;

Var Info : TSearchRec;
Count : Longint;

Begin

Count:=0;

If `FindFirst` ('*',faAnyFile **and** faDirectory ,Info)=0 **then**
begin

Repeat

Inc(Count);

With Info **do**

begin

If (Attr **and** faDirectory) = faDirectory **then**

Write ('Dir : ');

WriteLn (Name:40,Size:15);

end;

Until `FindNext`(info)<>0;

end;

FindClose(Info);

WriteLn ('Finished search. Found ',Count,' matches');

End.

FindNext

Synopsis: Find the next entry in a findhandle.

Declaration: function `FindNext`(var `Rslt`: TSearchRec) : LongInt

Visibility: default

Description: `FindNext` finds a next occurrence of a search sequence initiated by `FindFirst`. If another record matching the criteria in `Rslt` is found, 0 is returned, a nonzero constant is returned otherwise.

Remark: The last `FindNext` call must *always* be followed by a `FindClose` call with the same `Rslt` record. Failure to do so will result in memory loss.

For an example, see `FindFirst` (1134)

Errors: On error (no more file is found), a nonzero constant is returned.

See also: [FindFirst \(1134\)](#), [FindClose \(1134\)](#)

FloattoCurr

Synopsis: Convert a float to a Currency value.

Declaration: `function FloattoCurr(const Value: Extended) : Currency`

Visibility: default

Description: `FloatToCurr` converts the `Value` floating point value to a `Currency` value. It checks whether `Value` is in the valid range of currencies (determined by [MinCurrency \(1080\)](#) and [MaxCurrency \(1080\)](#)). If not, an [EConvertError \(1194\)](#) exception is raised.

Errors: If `Value` is out of range, an [EConvertError \(1194\)](#) exception is raised.

See also: [EConvertError \(1194\)](#), [TryFloatToCurr \(1192\)](#), [MinCurrency \(1080\)](#), [MaxCurrency \(1080\)](#)

FloatToDateTime

Synopsis: Convert a float to a `TDateTime` value.

Declaration: `function FloatToDateTime(const Value: Extended) : TDateTime`

Visibility: default

Description: `FloatToDateTime` converts the `Value` floating point value to a `TDateTime` value. It checks whether `Value` is in the valid range of dates (determined by [MinDateTime \(1080\)](#) and [MaxDateTime \(1080\)](#)). If not, an [EConvertError \(1194\)](#) exception is raised.

Errors: If `Value` is out of range, an [EConvertError \(1194\)](#) exception is raised.

See also: [EConvertError \(1194\)](#), [MinDateTime \(1080\)](#), [MaxDateTime \(1080\)](#)

FloatToDecimal

Synopsis: Convert a float value to a `TFloatRec` value.

Declaration: `procedure FloatToDecimal(var Result: TFloatRec; Value: Extended;
Precision: Integer; Decimals: Integer)`

Visibility: default

Description: `FloatToDecimal` converts the float `Value` to a float description in the `ResultTFloatRec (1084)` format. It will store `Precision` digits in the `Digits` field, of which at most `Decimal` decimals.

Errors: None.

See also: [TFloatRec \(1084\)](#)

FloatToStr

Synopsis: Convert a float value to a string using a fixed format.

Declaration: `function FloatToStr(Value: Extended) : String`

Visibility: default

Description: `FloatToStr` converts the floating point variable `Value` to a string representation. It will choose the shortest possible notation of the two following formats:

Fixed format will represent the string in fixed notation,

Decimal format will represent the string in scientific notation.

More information on these formats can be found in `FloatToStrF` (1137). `FloatToStr` is completely equivalent to the following call:

```
FloatToStrF(Value, ffGeneral, 15, 0);
```

Errors: None.

See also: `FloatToStrF` (1137), `FormatFloat` (1147), `StrToFloat` (1182)

Listing: `./sysutex/ex67.pp`

Program `Example67`;

{ This program demonstrates the FloatToStr function }

Uses `sysutils`;

Procedure `Testit` (`Value` : `Extended`);

begin

WriteIn (`Value`, ' -> ', `FloatToStr`(`Value`));

WriteIn (`-Value`, ' -> ', `FloatToStr`(`-Value`));

end;

Begin

`Testit` (0.0);

`Testit` (1.1);

`Testit` (1.1e-3);

`Testit` (1.1e-20);

`Testit` (1.1e-200);

`Testit` (1.1e+3);

`Testit` (1.1e+20);

`Testit` (1.1e+200);

End.

FloatToStrF

Synopsis: Convert a float value to a string using a given format.

Declaration: `function FloatToStrF(Value: Extended; format: TFloatFormat;
 Precision: Integer; Digits: Integer) : String`

Visibility: default

Description: `FloatToStrF` converts the floating point number value to a string representation, according to the settings of the parameters `Format`, `Precision` and `Digits`.

The meaning of the `Precision` and `Digits` parameter depends on the `Format` parameter. The format is controlled mainly by the `Format` parameter. It can have one of the following values:

ffcurrency Money format. Value is converted to a string using the global variables `CurrencyString`, `CurrencyFormat` and `NegCurrencyFormat`. The `Digits` parameter specifies the number of digits following the decimal point and should be in the range -1 to 18. If `Digits` equals -1, `CurrencyDecimals` is assumed. The `Precision` parameter is ignored.

ffExponent Scientific format. Value is converted to a string using scientific notation: 1 digit before the decimal point, possibly preceded by a minus sign if Value is negative. The number of digits after the decimal point is controlled by `Precision` and must lie in the range 0 to 15.

ffFixed Fixed point format. Value is converted to a string using fixed point notation. The result is composed of all digits of the integer part of Value, preceded by a minus sign if Value is negative. Following the integer part is `DecimalSeparator` and then the fractional part of Value, rounded off to `Digits` numbers. If the number is too large then the result will be in scientific notation.

ffGeneral General number format. The argument is converted to a string using `ffExponent` or `ffFixed` format, depending on which one gives the shortest string. There will be no trailing zeroes. If Value is less than 0.00001 or if the number of decimals left of the decimal point is larger than `Precision` then scientific notation is used, and `Digits` is the minimum number of digits in the exponent. Otherwise `Digits` is ignored.

ffnumber Is the same as `ffFixed`, except that thousand separators are inserted in the result string.

Errors: None.

See also: `FloatToStr` ([1137](#)), `FloatToText` ([1139](#))

Listing: `./sysutex/ex68.pp`

Program `Example68;`

```
{ This program demonstrates the FloatToStrF function }
```

```
Uses sysutils;
```

```
Const Fmt : Array [TFloatFormat] of string[10] =
    ( 'general', 'exponent', 'fixed', 'number', 'Currency' );
```

```
Procedure Testit (Value : Extended);
```

```
Var I, J : longint;
    FF : TFloatFormat;
```

```
begin
```

```
  For I:=5 to 15 do
```

```
    For J:=1 to 4 do
```

```
      For FF:=ffgeneral to ffcurrency do
```

```
        begin
```

```
          Write (Value, '(Prec: ', I:2, ', ', Dig: ', J, ', ', fmt : ', Fmt[ff], ') : ');
```

```
          Writeln (FloatToStrF(Value, FF, I, J));
```

```
          Write (-Value, '(Prec: ', I:2, ', ', Dig: ', J, ', ', fmt : ', Fmt[ff], ') : ');
```

```
          Writeln (FloatToStrF(-Value, FF, I, J));
```

```
        end;
```

```
end;
```

Begin

```

Testit (1.1);
Testit (1.1E1);
Testit (1.1E-1);
Testit (1.1E5);
Testit (1.1E-5);
Testit (1.1E10);
Testit (1.1E-10);
Testit (1.1E15);
Testit (1.1E-15);
Testit (1.1E100);
Testit (1.1E-100);

```

End.**FloatToText**

Synopsis: Return a string representation of a float, with a given format.

Declaration: `function FloatToText(Buffer: PChar;Value: Extended;format: TFloatFormat;
Precision: Integer;Digits: Integer) : LongInt`

Visibility: default

Description: `FloatToText` converts the floating point variable `Value` to a string representation and stores it in `Buffer`. The conversion is governed by `format`, `Precisison` and `Digits`. more information on these parameters can be found in `FloatToStrF` ([1137](#)). `Buffer` should point to enough space to hold the result. No checking on this is performed.

The result is the number of characters that was copied in `Buffer`.

Errors: None.

See also: `FloatToStr` ([1137](#)), `FloatToStrF` ([1137](#))

Listing: `./sysutex/ex69.pp`

Program Example68;

{ This program demonstrates the FloatToStrF function }

Uses sysutils;

Const Fmt : **Array** [TFloatFormat] **of** **string**[10] =
('general', 'exponent', 'fixed', 'number', 'Currency');

Procedure Testit (Value : Extended);

Var I, J : longint;
FF : TFloatFormat;
S : ShortString;

begin

For I:=5 to 15 do

For J:=1 to 4 do

For FF:=ffgeneral to ffcurrency do

begin

Write (Value, '(Prec: ', I:2, ', Dig: ', J, ', fmt : ', Fmt[ff], ') : ');


```

    SetLength(S, FloatToText (@S[1], Value, FF, I, J));
    Writeln (S);
    Write (-Value, '(Prec: ', I:2, ', Dig: ', J, ', fmt: ', Fmt[ff], ') : ');
    SetLength(S, FloatToText (@S[1], -Value, FF, I, J));
    Writeln (S);
end;

Begin
    Testit (1.1);
    Testit (1.1E1);
    Testit (1.1E-1);
    Testit (1.1E5);
    Testit (1.1E-5);
    Testit (1.1E10);
    Testit (1.1E-10);
    Testit (1.1E15);
    Testit (1.1E-15);
    Testit (1.1E100);
    Testit (1.1E-100);
End.

```

FloatToTextFmt

Synopsis: Convert a float value to a string using a given mask.

Declaration: `function FloatToTextFmt(Buffer: PChar; Value: Extended; format: PChar)
: Integer`

Visibility: default

Description: `FloatToTextFmt` returns a textual representation of `Value` in the memory location pointed to by `Buffer`. it uses the formatting specification in `Format` to do this. The return value is the number of characters that were written in the buffer.

For a list of valid formatting characters, see `FormatFloat` ([1147](#))

Errors: No length checking is performed on the buffer. The buffer should point to enough memory to hold the complete string. If this is not the case, an access violation may occur.

See also: `FormatFloat` ([1147](#))

FmtStr

Synopsis: Format a string with given arguments.

Declaration: `procedure FmtStr(var Res: String; const Fmt: String;
const args: Array[] of const)`

Visibility: default

Description: `FmtStr` calls `Format` ([1141](#)) with `Fmt` and `Args` as arguments, and stores the result in `Res`. For more information on how the resulting string is composed, see `Format` ([1141](#)).

Errors: In case of error, a `EConvertError` exception is raised.

See also: `Format` ([1141](#)), `FormatBuf` ([1146](#))

Listing: ./sysutex/ex70.pp

Program Example70;

{ This program demonstrates the FmtStr function }

Uses sysutils;

Var S : AnsiString;

Begin

S:= '';

FmtStr (S, 'For some nice examples of fomatting see %s.', ['Format']);

WriteLn (S);

End.

Format

Synopsis: Format a string with given arguments.

Declaration: function Format(const Fmt: String; const Args: Array[] of const) : String

Visibility: default

Description: Format replaces all placeholders in Fmt with the arguments passed in Args and returns the resulting string. A placeholder looks as follows:

```
'%' [Index':' ] ['-'] [Width] ['. ' Precision] ArgType
```

elements between single quotes must be typed as shown without the quotes, and elements between square brackets [] are optional. The meaning of the different elements is shown below:

'%' starts the placeholder. If you want to insert a literal % character, then you must insert two of them : %%.

Index ':' takes the Index-th element in the argument array as the element to insert.

'-' tells Format to left-align the inserted text. The default behaviour is to right-align inserted text. This can only take effect if the Width element is also specified.

Width the inserted string must have at least have Width characters. If not, the inserted string will be padded with spaces. By default, the string is left-padded, resulting in a right-aligned string. This behaviour can be changed by the '-' character.

'.' Precision Indicates the precision to be used when converting the argument. The exact meaning of this parameter depends on ArgType.

The Index, Width and Precision parameters can be replaced by *, in which case their value will be read from the next element in the Args array. This value must be an integer, or an EConvertError exception will be raised.

The argument type is determined from ArgType. It can have one of the following values (case insensitive):

D Decimal format. The next argument in the Args array should be an integer. The argument is converted to a decimal string,. If precision is specified, then the string will have at least Precision digits in it. If needed, the string is (left) padded with zeroes.

EScientific format. The next argument in the `Args` array should be a Floating point value. The argument is converted to a decimal string using scientific notation, using `FloatToStrF` (1137), where the optional precision is used to specify the total number of decimals. (default a value of 15 is used). The exponent is formatted using maximally 3 digits.

In short, the `E` specifier formats it's argument as follows:

```
FloatToStrF(Argument, ffExponent, Precision, 3)
```

FFixed point format. The next argument in the `Args` array should be a floating point value. The argument is converted to a decimal string, using fixed notation (see `FloatToStrF` (1137)). `Precision` indicates the number of digits following the decimal point.

In short, the `F` specifier formats it's argument as follows:

```
FloatToStrF(Argument, ffFixed, fixed, 9999, Precision)
```

GGeneral number format. The next argument in the `Args` array should be a floating point value. The argument is converted to a decimal string using fixed point notation or scientific notation, depending on which gives the shortest result. `Precision` is used to determine the number of digits after the decimal point.

In short, the `G` specifier formats it's argument as follows:

```
FloatToStrF(Argument, ffGeneral, Precision, 3)
```

MCurrency format. the next argument in the `var{Args}` array must be a floating point value. The argument is converted to a decimal string using currency notation. This means that fixed-point notation is used, but that the currency symbol is appended. If precision is specified, then then it overrides the `CurrencyDecimals` global variable used in the `FloatToStrF` (1137)

In short, the `M` specifier formats it's argument as follows:

```
FloatToStrF(Argument, ffCurrency, 9999, Precision)
```

NNumber format. This is the same as fixed point format, except that thousand separators are inserted in the resulting string.

PPointer format. The next argument in the `Args` array must be a pointer (typed or untyped). The pointer value is converted to a string of length 8, representing the hexadecimal value of the pointer.

SString format. The next argument in the `Args` array must be a string. The argument is simply copied to the result string. If `Precision` is specified, then only `Precision` characters are copied to the result string.

Xhexadecimal format. The next argument in the `Args` array must be an integer. The argument is converted to a hexadecimal string with just enough characters to contain the value of the integer. If `Precision` is specified then the resulting hexadecimal representation will have at least `Precision` characters in it (with a maximum value of 32).

Errors: In case of error, an `EConversionError` exception is raised. Possible errors are:

- 1.Errors in the format specifiers.
- 2.The next argument is not of the type needed by a specifier.
- 3.The number of arguments is not sufficient for all format specifiers.

See also: `FormatBuf` (1146)

Listing: `./sysutex/ex71.pp`

Program example71;

{ \$mode objfpc }

{ This program demonstrates the Format function }

Uses sysutils;

Var P : Pointer;
 fmt,S : **string**;

Procedure TestInteger;

begin

Try

 Fmt:= '[%d]'; S:=**Format** (Fmt,[10]); **writeln** (Fmt:12, ' => ',s);
 Fmt:= '[%%]'; S:=**Format** (Fmt,[10]); **writeln** (Fmt:12, ' => ',s);
 Fmt:= '[%10d]'; S:=**Format** (Fmt,[10]); **writeln** (Fmt:12, ' => ',s);
 fmt:= '[%.4d]'; S:=**Format** (fmt,[10]); **writeln** (Fmt:12, ' => ',s);
 Fmt:= '[%10.4d]'; S:=**Format** (Fmt,[10]); **writeln** (Fmt:12, ' => ',s);
 Fmt:= '[%0:d]'; S:=**Format** (Fmt,[10]); **writeln** (Fmt:12, ' => ',s);
 Fmt:= '[%0:10d]'; S:=**Format** (Fmt,[10]); **writeln** (Fmt:12, ' => ',s);
 Fmt:= '[%0:10.4d]'; S:=**Format** (Fmt,[10]); **writeln** (Fmt:12, ' => ',s);
 Fmt:= '[%0:-10d]'; S:=**Format** (Fmt,[10]); **writeln** (Fmt:12, ' => ',s);
 Fmt:= '[%0:-10.4d]'; S:=**Format** (fmt,[10]); **writeln** (Fmt:12, ' => ',s);
 Fmt:= '[%-*.d]'; S:=**Format** (fmt,[4,5,10]); **writeln** (Fmt:12, ' => ',s);

except

On E : **Exception do**

begin

Writeln ('Exception caught : ',E.Message);

end;

end;

writeln ('Press enter');

readln;

end;

Procedure TestHexadecimal;

begin

try

 Fmt:= '[%x]'; S:=**Format** (Fmt,[10]); **writeln** (Fmt:12, ' => ',s);
 Fmt:= '[%10x]'; S:=**Format** (Fmt,[10]); **writeln** (Fmt:12, ' => ',s);
 Fmt:= '[%10.4x]'; S:=**Format** (Fmt,[10]); **writeln** (Fmt:12, ' => ',s);
 Fmt:= '[%0:x]'; S:=**Format** (Fmt,[10]); **writeln** (Fmt:12, ' => ',s);
 Fmt:= '[%0:10x]'; S:=**Format** (Fmt,[10]); **writeln** (Fmt:12, ' => ',s);
 Fmt:= '[%0:10.4x]'; S:=**Format** (Fmt,[10]); **writeln** (Fmt:12, ' => ',s);
 Fmt:= '[%0:-10x]'; S:=**Format** (Fmt,[10]); **writeln** (Fmt:12, ' => ',s);
 Fmt:= '[%0:-10.4x]'; S:=**Format** (fmt,[10]); **writeln** (Fmt:12, ' => ',s);
 Fmt:= '[%-*.x]'; S:=**Format** (fmt,[4,5,10]); **writeln** (Fmt:12, ' => ',s);

except

On E : **Exception do**

begin

Writeln ('Exception caught : ',E.Message);

end;

end;

writeln ('Press enter');

readln;

end;

Procedure TestPointer;

```
begin
  P:=Pointer(1234567);
  try
    Fmt:='[0x%p]';S:=Format (Fmt,[P]);writeln(Fmt:12,'=>',s);
    Fmt:='[0x%10p]';S:=Format (Fmt,[P]);writeln(Fmt:12,'=>',s);
    Fmt:='[0x%10.4p]';S:=Format (Fmt,[P]);writeln(Fmt:12,'=>',s);
    Fmt:='[0x%0:p]';S:=Format (Fmt,[P]);writeln(Fmt:12,'=>',s);
    Fmt:='[0x%0:10p]';S:=Format (Fmt,[P]);writeln(Fmt:12,'=>',s);
    Fmt:='[0x%0:10.4p]';S:=Format (Fmt,[P]);writeln(Fmt:12,'=>',s);
    Fmt:='[0x%0:-10p]';S:=Format (Fmt,[P]);writeln(Fmt:12,'=>',s);
    Fmt:='[0x%0:-10.4p]';S:=Format (fmt,[P]);writeln(Fmt:12,'=>',s);
    Fmt:='[%-*.p]';S:=Format (fmt,[4,5,P]);writeln(Fmt:12,'=>',s);
  except
    On E : Exception do
      begin
        Writeln ('Exception caught : ',E.Message);
      end;
    end;
    writeln ('Press enter');
    readln;
  end;
```

Procedure TestString;

```
begin
  try
    Fmt:='[%s]';S:=Format(fmt,['This is a string']);Writeln(fmt:12,'=>',s);
    fmt:='[%0:s]';s:=Format(fmt,['This is a string']);Writeln(fmt:12,'=>',s);
    fmt:='[%0:18s]';s:=Format(fmt,['This is a string']);Writeln(fmt:12,'=>',s);
    fmt:='[%0:-18s]';s:=Format(fmt,['This is a string']);Writeln(fmt:12,'=>',s);
    fmt:='[%0:18.12s]';s:=Format(fmt,['This is a string']);Writeln(fmt:12,'=>',s);
    fmt:='[%-*.s]';s:=Format(fmt,[18,12,'This is a string']);Writeln(fmt:12,'=>',s);
  except
    On E : Exception do
      begin
        Writeln ('Exception caught : ',E.Message);
      end;
    end;
    writeln ('Press enter');
    readln;
  end;
```

Procedure TestExponential;

```
begin
  Try
    Fmt:='[%e]';S:=Format (Fmt,[1.234]);writeln(Fmt:12,'=>',s);
    Fmt:='[%10e]';S:=Format (Fmt,[1.234]);writeln(Fmt:12,'=>',s);
    Fmt:='[%10.4e]';S:=Format (Fmt,[1.234]);writeln(Fmt:12,'=>',s);
    Fmt:='[%0:e]';S:=Format (Fmt,[1.234]);writeln(Fmt:12,'=>',s);
    Fmt:='[%0:10e]';S:=Format (Fmt,[1.234]);writeln(Fmt:12,'=>',s);
    Fmt:='[%0:10.4e]';S:=Format (Fmt,[1.234]);writeln(Fmt:12,'=>',s);
    Fmt:='[%0:-10e]';S:=Format (Fmt,[1.234]);writeln(Fmt:12,'=>',s);
    Fmt:='[%0:-10.4e]';S:=Format (fmt,[1.234]);writeln(Fmt:12,'=>',s);
    Fmt:='[%-*.e]';S:=Format (fmt,[4,5,1.234]);writeln(Fmt:12,'=>',s);
```

```

except
  On E : Exception do
    begin
      WriteLn ( 'Exception caught : ',E.Message);
    end;
  end;
  writeln ( 'Press enter');
  readln;
end;

```

Procedure TestNegativeExponential;

```

begin
  Try
    Fmt:='[%e]';S:=Format (Fmt,[ -1.234]);writeln (Fmt:12,' => ',s);
    Fmt:='[%10e]';S:=Format (Fmt,[ -1.234]);writeln (Fmt:12,' => ',s);
    Fmt:='[%10.4e]';S:=Format (Fmt,[ -1.234]);writeln (Fmt:12,' => ',s);
    Fmt:='[%0:e]';S:=Format (Fmt,[ -1.234]);writeln (Fmt:12,' => ',s);
    Fmt:='[%0:10e]';S:=Format (Fmt,[ -1.234]);writeln (Fmt:12,' => ',s);
    Fmt:='[%0:10.4e]';S:=Format (Fmt,[ -1.234]);writeln (Fmt:12,' => ',s);
    Fmt:='[%0:-10e]';S:=Format (Fmt,[ -1.234]);writeln (Fmt:12,' => ',s);
    Fmt:='[%0:-10.4e]';S:=Format (Fmt,[ -1.234]);writeln (Fmt:12,' => ',s);
    Fmt:='[%-*.e]';S:=Format (Fmt,[4,5, -1.234]);writeln (Fmt:12,' => ',s);
  except
    On E : Exception do
      begin
        WriteLn ( 'Exception caught : ',E.Message);
      end;
    end;
    writeln ( 'Press enter');
    readln;
  end;

```

Procedure TestSmallExponential;

```

begin
  Try
    Fmt:='[%e]';S:=Format (Fmt,[0.01234]);writeln (Fmt:12,' => ',s);
    Fmt:='[%10e]';S:=Format (Fmt,[0.01234]);writeln (Fmt:12,' => ',s);
    Fmt:='[%10.4e]';S:=Format (Fmt,[0.01234]);writeln (Fmt:12,' => ',s);
    Fmt:='[%0:e]';S:=Format (Fmt,[0.01234]);writeln (Fmt:12,' => ',s);
    Fmt:='[%0:10e]';S:=Format (Fmt,[0.01234]);writeln (Fmt:12,' => ',s);
    Fmt:='[%0:10.4e]';S:=Format (Fmt,[0.01234]);writeln (Fmt:12,' => ',s);
    Fmt:='[%0:-10e]';S:=Format (Fmt,[0.0123]);writeln (Fmt:12,' => ',s);
    Fmt:='[%0:-10.4e]';S:=Format (Fmt,[0.01234]);writeln (Fmt:12,' => ',s);
    Fmt:='[%-*.e]';S:=Format (Fmt,[4,5,0.01234]);writeln (Fmt:12,' => ',s);
  except
    On E : Exception do
      begin
        WriteLn ( 'Exception caught : ',E.Message);
      end;
    end;
    writeln ( 'Press enter');
    readln;
  end;

```

Procedure TestSmallNegExponential;

```

begin
  Try
    Fmt:= '%e'; S:=Format (Fmt,[ -0.01234]); writeln (Fmt:12, ' => ',s);
    Fmt:= '%10e'; S:=Format (Fmt,[ -0.01234]); writeln (Fmt:12, ' => ',s);
    Fmt:= '%10.4e'; S:=Format (Fmt,[ -0.01234]); writeln (Fmt:12, ' => ',s);
    Fmt:= '%0:e'; S:=Format (Fmt,[ -0.01234]); writeln (Fmt:12, ' => ',s);
    Fmt:= '%0:10e'; S:=Format (Fmt,[ -0.01234]); writeln (Fmt:12, ' => ',s);
    Fmt:= '%0:10.4e'; S:=Format (Fmt,[ -0.01234]); writeln (Fmt:12, ' => ',s);
    Fmt:= '%0:-10e'; S:=Format (Fmt,[ -0.01234]); writeln (Fmt:12, ' => ',s);
    Fmt:= '%0:-10.4e'; S:=Format (Fmt,[ -0.01234]); writeln (Fmt:12, ' => ',s);
    Fmt:= '%-*.e'; S:=Format (Fmt,[4,5, -0.01234]); writeln (Fmt:12, ' => ',s);
  except
    On E : Exception do
      begin
        Writeln ( 'Exception caught : ',E.Message);
      end;
    end;
  writeln ( 'Press enter ');
  readln;
end;

begin
  TestInteger;
  TestHexadecimal;
  TestPointer;
  TestExponential;
  TestNegativeExponential;
  TestSmallExponential;
  TestSmallNegExponential;
  teststring;
end.

```

FormatBuf

Synopsis: Format a string with given arguments and store the result in a buffer.

Declaration: `function FormatBuf(var Buffer; BufLen: Cardinal; const Fmt;
 fmtLen: Cardinal; const Args: Array[] of const)
 : Cardinal`

Visibility: default

Description: FormatBuf calls formatfloat with Buf.

Errors:

Listing: ./sysutex/ex72.pp

Program Example72;

{ This program demonstrates the FormatBuf function }

Uses sysutils;

Var

S : ShortString;

```

Const
  Fmt : ShortString = 'For some nice examples of fomatting see %s.';

Begin
  S:= '';
  SetLength(S, FormatBuf (S[1], 255, Fmt[1], Length(Fmt), [ 'Format' ]));
  WriteLn (S);
End.

```

FormatDateTime

Synopsis: Return a string representation of a TDateTime value with a given format.

Declaration: function FormatDateTime(FormatStr: String; DateTime: TDateTime) : String

Visibility: default

Description: FormatDateTime formats the date and time encoded in DateTime according to the formatting given in FormatStr. The complete list of formatting characters can be found in [formatchars \(1074\)](#).

Errors: On error (such as an invalid character in the formatting string), and EConvertError exception is raised.

See also: DateTimeToStr ([1111](#)), DateToStr ([1113](#)), TimeToStr ([1189](#)), StrToDateTime ([1181](#))

Listing: ./sysutex/ex14.pp

Program Example14;

{ This program demonstrates the FormatDateTime function }

Uses sysutils;

Var ThisMoment : TDateTime;

```

Begin
  ThisMoment:=Now;
  WriteLn ( 'Now : ', FormatDateTime( 'hh:nn ', ThisMoment ));
  WriteLn ( 'Now : ', FormatDateTime( 'DD MM YYYY', ThisMoment ));
  WriteLn ( 'Now : ', FormatDateTime( 'c ', ThisMoment ));
End.

```

FormatFloat

Synopsis: Format a float according to a certain mask.

Declaration: function FormatFloat(const Format: String; Value: Extended) : String

Visibility: default

Description: FormatFloat formats the floating-point value given by Value using the format specifications in Format. The format specifier can give format specifications for positive, negative or zero values (separated by a semicolon).

If the formatspecifier is empty or the value needs more than 18 digits to be correctly represented, the result is formatted with a call to FloatToStrF ([1137](#)) with the ffGeneral format option.

The following format specifiers are supported:

0 is a digit place holder. If there is a corresponding digit in the value being formatted, then it replaces the 0. If not, the 0 is left as-is.

is also a digit place holder. If there is a corresponding digit in the value being formatted, then it replaces the #. If not, it is removed. by a space.

. determines the location of the decimal point. Only the first '.' character is taken into account. If the value contains digits after the decimal point, then it is replaced by the value of the `DecimalSeparator` character.

, determines the use of the thousand separator character in the output string. If the format string contains one or more ',' characters, then thousand separators will be used. The `ThousandSeparator` character is used.

E+ determines the use of scientific notation. If 'E+' or 'E-' (or their lowercase counterparts) are present then scientific notation is used. The number of digits in the output string is determined by the number of 0 characters after the 'E+'.

; This character separates sections for positive, negative, and zero numbers in the format string.

Errors: If an error occurs, an exception is raised.

See also: `FloatToStr` ([1137](#))

Listing: `./sysutex/ex89.pp`

Program `Example89;`

```
{ This program demonstrates the FormatFloat function }
```

Uses `sysutils;`

Const

```
NrFormat=9;
FormatStrings : Array[1..NrFormat] of string = (
    ' ',
    '0',
    '0.00',
    '#.##',
    '#.##0.00',
    '#.##0.00;(#.##0.00)',
    '#.##0.00;;Zero',
    '0.000E+00',
    '#.###E-0');
```

```
NrValue = 5;
FormatValues : Array[1..NrValue] of Double =
    (1234, -1234, 0.5, 0, -0.5);
```

```
Width  = 12;
FWidth = 20;
```

Var

```
I, J : Integer;
S : String;
```

begin

```
Write('Format': FWidth);
For I:=1 to NrValue do
    Write(FormatValues[I]: Width:2);
WriteLn;
For I:=1 to NrFormat do
```

```

begin
Write (FormatStrings [ i ] : FWidth);
For J:=1 to NrValue do
begin
S:=FormatFloat (FormatStrings [ I ], FormatValues [ j ]);
Write (S : Width);
end;
Writeln;
end;
End.

```

FreeAndNil

Synopsis: Free object if needed, and set object reference to Nil

Declaration: `procedure FreeAndNil (var obj)`

Visibility: default

Description: `FreeAndNil` will free the object in `Obj` and will set the reference in `Obj` to `Nil`. The reference is set to `Nil` first, so if an exception occurs in the destructor of the object, the reference will be `Nil` anyway.

Errors: Exceptions that occur during the destruction of `Obj` are not caught.

GetAppConfigDir

Synopsis: Return the appropriate directory for the application's configuration files.

Declaration: `function GetAppConfigDir (Global: Boolean) : String`

Visibility: default

Description: `GetAppConfigDir` returns the name of a directory in which the application should store its configuration files on the current OS. If the parameter `Global` is `True` then the directory returned is a global directory, i.e. valid for all users on the system. If the parameter `Global` is false, then the directory is specific for the user who is executing the program. On systems that do not support multi-user environments, these two directories may be the same.

The directory which is returned is the name of the directory where the application is supposed to store files. This does not mean that the directory exists, or that the user can write in this directory (especially if `Global=True`). It just returns the name of the appropriate location.

On systems where the operating system provides a call to determine this location, this call will be used. On systems where there is no such call, an algorithm is used which reflects common practice on that system.

The application name is deduced from the binary name via the `ApplicationName` (1103) call, but can be configured by means of the `OnGetApplicationName` (1087) callback.

Errors: None.

See also: `GetAppConfigFile` (1150), `ApplicationName` (1103), `OnGetApplicationName` (1087), `CreateDir` (1109), `SysConfigDir` (1081)

GetAppConfigFile

Synopsis: Return an appropriate name for an application configuration file.

Declaration: `function GetAppConfigFile(Global: Boolean) : String`
`function GetAppConfigFile(Global: Boolean;SubDir: Boolean) : String`

Visibility: default

Description: `GetAppConfigFile` returns the name of a file in which the application can store its configuration parameters. The `Global` parameter determines whether it is a global configuration file (value `True`) or a personal configuration file (value `False`). The parameter `SubDir`, in case it is set to `True`, will insert the name of a directory before the filename. This can be used in case the application needs to store other data than configuration data in an application-specific directory. Default behaviour is to set this to `False`.

No assumptions should be made about the existence or writeability of this file, or the directory where the file should reside.

On systems where the operating system provides a call to determine the location of configuration files, this call will be used. On systems where there is no such call, an algorithm is used which reflects common practice on that system.

The application name is deduced from the binary name via the `ApplicationName` (1103) call, but can be configured by means of the `OnGetApplicationName` (1087) callback.

Errors: None.

See also: `GetAppConfigDir` (1149), `OnGetApplicationName` (1087), `ApplicationName` (1103), `CreateDir` (1109), `ConfigExtension` (1076), `SysConfigDir` (1081)

GetCurrentDir

Synopsis: Return the current working directory of the application.

Declaration: `function GetCurrentDir : String`

Visibility: default

Description: `GetCurrentDir` returns the current working directory.

Errors: None.

See also: `SetCurrentDir` (1165), `DiskFree` (1117), `DiskSize` (1117)

Listing: `./sysutex/ex28.pp`

Program `Example28`;

{ This program demonstrates the `GetCurrentDir` function }

Uses `sysutils`;

Begin

`WriteLn ('Current Directory is : ',GetCurrentDir);`

End.

GetDirs

Synopsis: Return a list of directory names from a path.

Declaration: `function GetDirs(var DirName: String; var Dirs: Array[] of pchar)
: LongInt`

Visibility: default

Description: `GetDirs` splits `DirName` in a null-byte separated list of directory names, `Dirs` is an array of `PChars`, pointing to these directory names. The function returns the number of directories found, or -1 if none were found. `DirName` must contain only `OSDirSeparator` as Directory separator chars.

Errors: None.

See also: `ExtractRelativePath` ([1125](#))

Listing: `./sysutex/ex45.pp`

Program `Example45;`

```
{ This program demonstrates the GetDirs function }  
{ $H+ }
```

Uses `sysutils;`

```
Var Dirs : Array[0..127] of pchar;  
    I, Count : longint;  
    Dir, NewDir : String;
```

Begin

```
    Dir := GetCurrentDir;  
    Writeln ( 'Dir : ', Dir );  
    NewDir := '';  
    count := GetDirs ( Dir , Dirs );  
    For I := 0 to Count - 1 do  
        begin  
            NewDir := NewDir + ' / ' + StrPas ( Dirs [ I ] );  
            Writeln ( NewDir );  
        end;
```

End.

GetEnvironmentString

Synopsis: Return an environment variable by index.

Declaration: `function GetEnvironmentString(Index: Integer) : String`

Visibility: default

Description: `GetEnvironmentString` returns the `Index`-th environment variable. The index is 1 based, and is bounded from above by the result of `GetEnvironmentVariableCount` ([1152](#)).

For an example, `GetEnvironmentVariableCount` ([1152](#)).

Errors: If there is no environment, -1 may be returned.

See also: `GetEnvironmentVariable` ([1152](#)), `GetEnvironmentVariableCount` ([1152](#))

GetEnvironmentVariable

Synopsis: Return the value of an environment variable.

Declaration: `function GetEnvironmentVariable(const EnvVar: String) : String`

Visibility: default

Description: `GetEnvironmentVariable` returns the value of the `EnvVar` environment variable. If the specified variable does not exist or `EnvVar` is empty, an empty string is returned.

See also: `GetEnvironmentString` (1151), `GetEnvironmentVariableCount` (1152)

GetEnvironmentVariableCount

Synopsis: Return the number of variables in the environment.

Declaration: `function GetEnvironmentVariableCount : Integer`

Visibility: default

Description: `GetEnvironmentVariableCount` returns the number of variables in the environment. The number is 1 based, but the result may be zero if there are no environment variables.

Errors: If there is no environment, -1 may be returned.

See also: `GetEnvironmentString` (1151), `GetEnvironmentVariable` (1152)

Listing: ./sysutex/ex92.pp

```

{ $h+ }
program example92;

{ This program demonstrates the
  GetEnvironmentVariableCount function }

uses sysutils;

Var
  I : Integer;

begin
  For I:=1 to GetEnvironmentVariableCount do
    Writeln(i:3, ' : ', GetEnvironmentString(i));
end.

```

GetFileHandle

Synopsis: Extract OS handle from an untyped file or text file.

Declaration: `function GetFileHandle(var f: File) : LongInt`
`function GetFileHandle(var f: Text) : LongInt`

Visibility: default

Description: `GetFileHandle` returns the operating system handle for the file descriptor `F`. It can be used in various file operations which are not directly supported by the pascal language.

GetLastError

Synopsis: Return the last code from the OS.

Declaration: `function GetLastError : Integer`

Visibility: default

Description: `GetLastError` returns the error code from the last operating system call. It does not reset this code. In general, it should be called when an operating system call reported an error condition. In that case, `GetLastError` gives extended information about the error.

No assumptions should be made about the resetting of the error code by subsequent OS calls. This may be platform dependent.

See also: `RaiseLastError` ([1163](#))

GetLocalTime

Synopsis: Get the local time.

Declaration: `procedure GetLocalTime(var SystemTime: TSystemTime)`

Visibility: default

Description: `GetLocalTime` returns the system time in a `TSystemTime` ([1086](#)) format.

Errors: None.

See also: `Now` ([1162](#)), `Date` ([1110](#)), `Time` ([1188](#)), `TSystemTime` ([1086](#))

GetTempDir

Synopsis: Return name of system's temporary directory

Declaration: `function GetTempDir(Global: Boolean) : String`
`function GetTempDir : String`

Visibility: default

Description: `GetTempDir` returns the temporary directory of the system. If `Global` is `True` (the default value) it returns the system temporary directory, if it is `False` then a directory private to the user is returned. The returned name will end with a directory delimiter character.

These directories may be the same. No guarantee is made that this directory exists or is writeable by the user.

The `OnGetTempDir` ([1088](#)) handler may be set to provide custom handling of this routine: One could implement callbacks which take into consideration frameworks like KDE or GNOME, and return a different value from the default system implementation.

Errors: On error, an empty string is returned.

See also: `OnGetTempDir` ([1088](#)), `GetTempFileName` ([1154](#))

GetTempFileName

Synopsis: Return the name of a temporary file.

Declaration: `function GetTempFileName(const Dir: String;const Prefix: String)
: String
function GetTempFileName : String`

Visibility: default

Description: `GetTempFileName` returns the name of a temporary file in directory `Dir`. The name of the file starts with `Prefix`.

If `Dir` is empty, the value returned by `GetTempDir` is used, and if `Prefix` is empty, 'TMP' is used.

The `OnGetTempFile` (1088) handler may be set to provide custom handling of this routine: One could implement callbacks which take into consideration frameworks like KDE or GNOME, and return a different value from the default system implementation.

Errors: On error, an empty string is returned.

See also: `GetTempDir` (1153), `OnGetTempFile` (1088)

IncludeTrailingBackslash

Synopsis: Add trailing directory separator to a pathname, if needed.

Declaration: `function IncludeTrailingBackslash(const Path: String) : String`

Visibility: default

Description: `IncludeTrailingBackslash` is provided for backwards compatibility with Delphi. Use `IncludeTrailingPathDelimiter` (1154) instead.

See also: `IncludeTrailingPathDelimiter` (1154), `ExcludeTrailingPathDelimiter` (1121), `PathDelim` (1081), `IsPathDelimiter` (1158)

IncludeTrailingPathDelimiter

Synopsis: Add trailing directory separator to a pathname, if needed.

Declaration: `function IncludeTrailingPathDelimiter(const Path: String) : String`

Visibility: default

Description: `IncludeTrailingPathDelimiter` adds a trailing path delimiter character (`PathDelim` (1081)) to `Path` if none is present yet, and returns the result. If `Path` is empty, nothing is added.

See also: `IncludeTrailingBackslash` (1154), `ExcludeTrailingPathDelimiter` (1121), `PathDelim` (1081), `IsPathDelimiter` (1158)

IncMonth

Synopsis: Increases the month in a `TDateTime` value with a given amount.

Declaration: `function IncMonth(const DateTime: TDateTime;NumberOfMonths: Integer)
: TDateTime`

Visibility: default

Description: `IncMonth` increases the month number in `DateTime` with `NumberOfMonths`. It wraps the result as to get a month between 1 and 12, and updates the year accordingly. `NumberOfMonths` can be negative, and can be larger than 12 (in absolute value).

Errors: None.

See also: `Date` ([1110](#)), `Time` ([1188](#)), `Now` ([1162](#))

Listing: `./sysutex/ex15.pp`

Program `Example15`;

{ This program demonstrates the IncMonth function }

Uses `sysutils`;

Var `ThisDay` : `TDateTime`;

Begin

```
ThisDay:=Date;
Writeln ( 'ThisDay : ', DateToStr ( ThisDay ));
Writeln ( '6 months ago : ', DateToStr ( IncMonth ( ThisDay , -6)));
Writeln ( '6 months from now : ', DateToStr ( IncMonth ( ThisDay , 6)));
Writeln ( '12 months ago : ', DateToStr ( IncMonth ( ThisDay , -12)));
Writeln ( '12 months from now : ', DateToStr ( IncMonth ( ThisDay , 12)));
Writeln ( '18 months ago : ', DateToStr ( IncMonth ( ThisDay , -18)));
Writeln ( '18 months from now : ', DateToStr ( IncMonth ( ThisDay , 18)));
```

End.

InterLockedDecrement

Synopsis: Thread-safe integer decrement

Declaration: `function InterLockedDecrement (var Target: LongInt) : LongInt`

Visibility: default

Description: `InterlockedDecrement` decrements `Target` in a thread-safe way, and returns the new value of `Target`

See also: `InterlockedIncrement` ([1156](#)), `InterlockedExchange` ([1155](#)), `InterlockedExchangeAdd` ([1156](#))

InterLockedExchange

Synopsis: Thread-safe exchange of 2 values.

Declaration: `function InterLockedExchange (var Target: LongInt; Source: LongInt) : LongInt`

Visibility: default

Description: `InterlockedDecrement` replaces `Target` with `Source` in a thread-safe way, and returns the old value of `Target`

See also: `InterlockedIncrement` ([1156](#)), `InterlockedDecrement` ([1155](#)), `InterlockedExchange` ([1155](#)), `InterlockedExchangeAdd` ([1156](#))

InterLockedExchangeAdd

Synopsis: Thread-safe exchange of 2 values

Declaration: `function InterLockedExchangeAdd(var Target: LongInt; Source: LongInt)
: LongInt`

Visibility: default

Description: `InterlockedDecrement` adds to `Target` the value of `Source` in a thread-safe way, and returns the old value of `Target`

See also: `InterlockedIncrement` ([1156](#)), `InterlockedDecrement` ([1155](#)), `InterlockedExchange` ([1155](#))

InterLockedIncrement

Synopsis: Thread-safe integer increment.

Declaration: `function InterLockedIncrement(var Target: LongInt) : LongInt`

Visibility: default

Description: `InterlockedIncrement` increments `Target` in a thread-safe way, and returns the new value of `Target`

See also: `InterlockedDecrement` ([1155](#)), `InterlockedExchange` ([1155](#)), `InterlockedExchangeAdd` ([1156](#))

IntToHex

Synopsis: Convert an integer value to a hexadecimal string.

Declaration: `function IntToHex(Value: Integer; Digits: Integer) : String
function IntToHex(Value: Int64; Digits: Integer) : String`

Visibility: default

Description: `IntToHex` converts `Value` to a hexadecimal string representation. The result will contain at least `Digits` characters. If `Digits` is less than the needed number of characters, the string will NOT be truncated. If `Digits` is larger than the needed number of characters, the result is padded with zeroes.

Errors: None.

See also: `IntToStr` ([1157](#))

Listing: `./sysutex/ex73.pp`

Program `Example73;`

`{ This program demonstrates the IntToHex function }`

Uses `sysutils;`

Var `I : longint;`

Begin

`For I:=0 to 31 do`

`begin`

`WriteLn (IntToHex(1 shl I,8));`

```

    Writeln (IntToHex(15 shl I,8))
end;
End.

```

IntToStr

Synopsis: Convert an integer value to a decimal string.

Declaration: `function IntToStr(Value: Integer) : String`
`function IntToStr(Value: Int64) : String`
`function IntToStr(Value: QWord) : String`

Visibility: default

Description: `IntToStr` converts `Value` to its string representation. The resulting string has only as much characters as needed to represent the value. If the value is negative a minus sign is prepended to the string.

Errors: None.

See also: `IntToHex` ([1156](#)), `StrToInt` ([1183](#))

Listing: `./sysutex/ex74.pp`

Program `Example74`;

{ This program demonstrates the IntToStr function }

Uses `sysutils`;

Var `I` : `longint`;

Begin

For `I:=0 to 31 do`

begin

Writeln (`IntToStr(1 shl I)`);

Writeln (`IntToStr(15 shl I)`);

end;

End.

IsDelimiter

Synopsis: Check whether a given string is a delimiter character.

Declaration: `function IsDelimiter(const Delimiters: String;const S: String;`
`Index: Integer) : Boolean`

Visibility: default

Description: `IsDelimiter` checks whether the `Index`-th character in the string `S` is a delimiter character as passed in `Delimiters`. If `Index` is out of range, `False` is returned.

Errors: None.

See also: `LastDelimiter` ([1159](#))

IsLeapYear

Synopsis: Determine whether a year is a leap year.

Declaration: `function IsLeapYear(Year: Word) : Boolean`

Visibility: default

Description: `IsLeapYear` returns `True` if `Year` is a leap year, `False` otherwise.

Errors: None.

See also: `IncMonth` ([1154](#)), `Date` ([1110](#))

Listing: `./sysutex/ex16.pp`

Program `Example16;`

{ This program demonstrates the IsLeapYear function }

Uses `sysutils;`

Var `YY,MM,dd : Word;`

Procedure `TestYear (Y : Word);`

begin

`WriteLn (Y, ' is leap year : ',IsLeapYear(Y));`
end;

Begin

`DeCodeDate(Date ,YY,mm,dd);`
`TestYear(yy);`
`TestYear(2000);`
`TestYear(1900);`
`TestYear(1600);`
`TestYear(1992);`
`TestYear(1995);`

End.

IsPathDelimiter

Synopsis: Is the character at the given position a pathdelimiter ?

Declaration: `function IsPathDelimiter(const Path: String;Index: Integer) : Boolean`

Visibility: default

Description: `IsPathDelimiter` returns `True` if the character at position `Index` equals `PathDelim` ([1081](#)), i.e. if it is a path delimiter character for the current platform.

Errors: `IncludeTrailingPathDelimiter` ([1154](#))`ExcludeTrailingPathDelimiter` ([1121](#))`PathDelim` ([1081](#))

IsValidIdent

Synopsis: Check whether a string is a valid identifier name.

Declaration: `function IsValidIdent(const Ident: String) : Boolean`

Visibility: default

Description: `IsValidIdent` returns `True` if `Ident` can be used as a component name. It returns `False` otherwise. `Ident` must consist of a letter or underscore, followed by a combination of letters, numbers or underscores to be a valid identifier.

Errors: None.

Listing: ./sysutex/ex75.pp

Program Example75;

```
{ This program demonstrates the IsValidIdent function }
```

Uses sysutils;

```
Procedure Testit (S : String);
```

begin

Write (' " ' , S , ' " is ');

If not IsValidIdent(S) then

```
Write( 'NOT' );
```

```
WriteIn ( 'a valid identifier');
```

end ;

Begin

```
Testit ( '_MyObj' );
```

```
Testit ( 'My__Obj1 ' );
```

```
Testit ( 'My_1_Obj' );
```

```
Testit ( '1MyObject' );
```

```
Testit ( 'My@Object' );
```

```
Testit ( 'M123' );
```

End .

LastDelimiter

Synopsis: Return the last occurrence of a set of delimiters in a string.

```
Declaration: function LastDelimiter(const Delimiters: String;const S: String)
                                     : Integer
```

Visibility: default

Description: LastDelimiter returns the *last* occurrence of any character in the set Delimiters in the string S.

Errors:

Listing: ./sysutex/ex88.pp

Program example88;

```
{ This program demonstrates the LastDelimiter function }
```

```
uses SysUtils;
```

begin

```
WriteIn( LastDelimiter( '\.: ', 'c:\filename.ext' ));
```

end.

LeftStr

Synopsis: Return a number of characters starting at the left of a string.

Declaration: `function LeftStr(const S: String; Count: Integer) : String`

Visibility: default

Description: `LeftStr` returns the `Count` leftmost characters of `S`. It is equivalent to a call to `Copy (S , 1 , Count)`.

Errors: None.

See also: `RightStr` ([1164](#)), `TrimLeft` ([1190](#)), `TrimRight` ([1191](#)), `Trim` ([1190](#))

Listing: `./sysutex/ex76.pp`

Program `Example76;`

{ This program demonstrates the LeftStr function }

Uses `sysutils;`

Begin

`WriteLn (LeftStr ('abcdefghijklmnopqrstuvwxyz ' , 20));`

`WriteLn (LeftStr ('abcdefghijklmnopqrstuvwxyz ' , 15));`

`WriteLn (LeftStr ('abcdefghijklmnopqrstuvwxyz ' , 1));`

`WriteLn (LeftStr ('abcdefghijklmnopqrstuvwxyz ' , 200));`

End.

LoadStr

Synopsis: Load a string from the resource tables.

Declaration: `function LoadStr(Ident: Integer) : String`

Visibility: default

Description: This function is not yet implemented. resources are not yet supported.

Errors:

LowerCase

Synopsis: Return a lowercase version of a string.

Declaration: `function LowerCase(const s: String) : String; Overload`

Visibility: default

Description: `LowerCase` returns the lowercase equivalent of `S`. Ansi characters are not taken into account, only ASCII codes below 127 are converted. It is completely equivalent to the lowercase function of the system unit, and is provided for compatiibility only.

Errors: None.

See also: `AnsiLowerCase` ([1093](#)), `UpperCase` ([1193](#)), `AnsiUpperCase` ([1101](#))

Listing: `./sysutex/ex77.pp`

Program Example77;

{ This program demonstrates the LowerCase function }

Uses sysutils;

Begin

WriteLn (**LowerCase**('THIS WILL COME out all LoWeRcAsE !'));

End.

MSecsToTimeStamp

Synopsis: Convert a number of milliseconds to a TDateTime value.

Declaration: function MSecsToTimeStamp(MSecs: Comp) : TTimeStamp

Visibility: default

Description: MSecsToTimeStamp converts the given number of milliseconds to a TTimeStamp date/time notation.

Use TTimeStamp variables if you need to keep very precise track of time.

Errors: None.

See also: TimeStampToMSecs ([1189](#)), DateTimeToTimeStamp ([1113](#))

Listing: ./sysutex/ex17.pp

Program Example17;

{ This program demonstrates the MSecsToTimeStamp function }

Uses sysutils;

Var MS : Comp;

 TS : TTimeStamp;

 DT : TDateTime;

Begin

 TS:=DateTimeToTimeStamp(**Now**);

WriteLn ('Now in days since 1/1/0001 : ', TS.**Date**);

WriteLn ('Now in millisecs since midnight : ', TS.**Time**);

 MS:=TimeStampToMSecs(TS);

WriteLn ('Now in millisecs since 1/1/0001 : ', MS);

 MS:=MS-1000*3600*2;

 TS:=MSecsToTimeStamp(MS);

 DT:=TimeStampToDateTime(TS);

WriteLn ('Now minus 1 day : ', **DateTimeToStr**(DT));

End.

NewStr

Synopsis: Allocate a new ansistring on the heap.

Declaration: function NewStr(const S: String) : PString

Visibility: default

Description: `NewStr` assigns a new dynamic string on the heap, copies `S` into it, and returns a pointer to the newly assigned string.

This function is obsolete, and shouldn't be used any more. The `AnsiString` mechanism also allocates ansistrings on the heap, and should be preferred over this mechanism.

For an example, see `AssignStr` (1103).

Errors: If not enough memory is present, an `EOutOfMemory` exception will be raised.

See also: `AssignStr` (1103), `DisposeStr` (1118)

Now

Synopsis: Returns the current date and time.

Declaration: `function Now : TDateTime`

Visibility: default

Description: `Now` returns the current date and time. It is equivalent to `Date+Time`.

Errors: None.

See also: `Date` (1110), `Time` (1188)

Listing: `./sysutex/ex18.pp`

Program `Example18;`

{ This program demonstrates the Now function }

Uses `sysutils;`

Begin

`WriteLn ('Now : ', DateTimeToStr(Now));`

End.

OutOfMemoryError

Synopsis: Raise an `EOutOfMemory` exception

Declaration: `procedure OutOfMemoryError`

Visibility: default

Description: `OutOfMemoryError` raises an `EOutOfMemory` (1197) exception, with an exception object that has been allocated on the heap at program startup. The program should never create an `EOutOfMemory` (1197) exception, but always call this routine.

See also: `EOutOfMemory` (1197)

QuotedStr

Synopsis: Return a quotes version of a string.

Declaration: `function QuotedStr(const S: String) : String`

Visibility: default

Description: `QuotedStr` returns the string `S`, quoted with single quotes. This means that `S` is enclosed in single quotes, and every single quote in `S` is doubled. It is equivalent to a call to `AnsiQuotedStr(s, '"')`.

Errors: None.

See also: `AnsiQuotedStr` ([1094](#)), `AnsiExtractQuotedStr` ([1092](#))

Listing: `./sysutex/ex78.pp`

Program `Example78;`

{ This program demonstrates the QuotedStr function }

Uses `sysutils;`

Var `S : AnsiString;`

Begin

`S:= 'He said ''Hello'' and walked on';`

`Writeln (S);`

`Writeln (' becomes');`

`Writeln (QuotedStr(S));`

End.

RaiseLastError

Synopsis: Raise an exception with the last Operating System error code.

Declaration: `procedure RaiseLastError`

Visibility: default

Description: `RaiseLastError` raises an `EOSError` ([1197](#)) exception with the error code returned by `GetLastError`. If the Error code is nonzero, then the corresponding error message will be returned. If the error code is zero, a standard message will be returned.

Errors: This procedure may not be implemented on all platforms. If it is not, then a normal Exception ([1199](#)) will be raised.

See also: `EOSError` ([1197](#)), `GetLastError` ([1153](#)), Exception ([1199](#))

RemoveDir

Synopsis: Remove a directory from the filesystem.

Declaration: `function RemoveDir(const Dir: String) : Boolean`

Visibility: default

Description: RemoveDir removes directory Dir from the disk. If the directory is not absolute, it is appended to the current working directory.

For an example, see CreateDir (1109).

Errors: In case of error (e.g. the directory isn't empty) the function returns False. If successful, True is returned.

RenameFile

Synopsis: Rename a file.

Declaration: `function RenameFile(const OldName: String;const NewName: String)
: Boolean`

Visibility: default

Description: RenameFile renames a file from OldName to NewName. The function returns True if successful, False otherwise. *Remark:* you cannot rename across disks or partitions.

Errors: On Error, False is returned.

See also: DeleteFile (1116)

Listing: ./sysutex/ex44.pp

Program Example44;

{ This program demonstrates the RenameFile function }

Uses sysutils;

Var F : Longint;
S : **String**;

Begin

S:= 'Some short file .';
F:= FileCreate ('test.dap');
FileWrite (F,S[1],Length(S));
FileClose(F);
If RenameFile ('test.dap', 'test.dat') **then**
 WriteLn ('Successfully renamed files .');
End.

RightStr

Synopsis: Return a number of characters from a string, starting at the end.

Declaration: `function RightStr(const S: String;Count: Integer) : String`

Visibility: default

Description: RightStr returns the Count rightmost characters of S. It is equivalent to a call to Copy (S , Length (S) + 1 - Count , Count).
If Count is larger than the actual length of S only the real length will be used.

Errors: None.

See also: LeftStr (1160), Trim (1190), TrimLeft (1190), TrimRight (1191)

Listing: ./sysutex/ex79.pp

Program Example79;

{ This program demonstrates the RightStr function }

Uses sysutils;

Begin

WriteLn (RightStr('abcdefghijklmnopqrstuvwxyz',20));

WriteLn (RightStr('abcdefghijklmnopqrstuvwxyz',15));

WriteLn (RightStr('abcdefghijklmnopqrstuvwxyz',1));

WriteLn (RightStr('abcdefghijklmnopqrstuvwxyz',200));

End.

SameFileName

Synopsis: Are two filenames referring to the same file ?

Declaration: function SameFileName(const S1: String;const S2: String) : Boolean

Visibility: default

Description: SameFileName returns True if calling AnsiCompareFileName (1090) with arguments S1 and S2 returns 0, and returns False otherwise.

Errors: None.

See also: AnsiCompareFileName (1090)

SameText

Synopsis: Checks whether 2 strings are the same (case insensitive)

Declaration: function SameText(const s1: String;const s2: String) : Boolean

Visibility: default

Description: SameText calls CompareText (1108) with S1 and S2 as parameters and returns True if the result of that call is zero, or False otherwise.

Errors: None.

See also: CompareText (1108), AnsiSameText (1095), AnsiSameStr (1095)

SetCurrentDir

Synopsis: Set the current directory of the application.

Declaration: function SetCurrentDir(const NewDir: String) : Boolean

Visibility: default

Description: SetCurrentDir sets the current working directory of your program to NewDir. It returns True if the function was successful, False otherwise.

Errors: In case of error, False is returned.

See also: GetCurrentDir (1150)

SetDirSeparators

Synopsis: Set the directory separators to the known directory separators.

Declaration: `function SetDirSeparators(const FileName: String) : String`

Visibility: default

Description: `SetDirSeparators` returns `FileName` with all possible `DirSeparators` replaced by `OSDirSeparator`.

Errors: None.

See also: `ExpandFileName` ([1122](#)), `ExtractFilePath` ([1124](#)), `ExtractFileDir` ([1123](#))

Listing: `./sysutex/ex47.pp`

Program `Example47;`

{ This program demonstrates the SetDirSeparators function }

Uses `sysutils;`

Begin

WriteLn (`SetDirSeparators(' /pp\bin /win32\ppc386 ')`);

End.

ShowException

Synopsis: Show the current exception to the user.

Declaration: `procedure ShowException(ExceptObject: TObject; ExceptAddr: Pointer)`

Visibility: default

Description: `ShowException` shows a message stating that a `ExceptObject` was raised at address `ExceptAddr`.

It uses `ExceptionErrorMessage` ([1120](#)) to create the message, and is aware of the fact whether the application is a console application or a GUI application. For a console application, the message is written to standard error output. For a GUI application, `OnShowException` ([1088](#)) is executed.

Errors: If, for a GUI application, `OnShowException` ([1088](#)) is not set, no message will be displayed to the user.

The exception message can be at most 255 characters long: It is possible that no memory can be allocated on the heap, so ansistrings are not available, so a shortstring is used to display the message.

See also: `ExceptObject` ([1121](#)), `ExceptAddr` ([1120](#)), `ExceptionErrorMessage` ([1120](#))

Sleep

Synopsis: Suspend execution of a program for a certain time.

Declaration: `procedure Sleep(milliseconds: Cardinal)`

Visibility: default

Description: `Sleep` suspends the execution of the program for the specified number of milliseconds (`milliseconds`).

After the specified period has expired, program execution resumes.

Remark: The indicated time is not exact, i.e. it is a minimum time. No guarantees are made as to the exact duration of the suspension.

StrAlloc

Synopsis: Allocate a null-terminated string on the heap.

Declaration: `function StrAlloc(Size: cardinal) : PChar`

Visibility: default

Description: `StrAlloc` reserves memory on the heap for a string with length `Len`, terminating `#0` included, and returns a pointer to it.

Additionally, `StrAlloc` allocates 4 extra bytes to store the size of the allocated memory. Therefore this function is NOT compatible with the `StrAlloc` (1167) function of the `Strings` unit.

For an example, see `StrBufSize` (1167).

Errors: None.

See also: `StrBufSize` (1167), `StrDispose` (1169), `StrAlloc` (1167)

StrBufSize

Synopsis: Return the size of a null-terminated string allocated on the heap.

Declaration: `function StrBufSize(Str: PChar) : SizeUInt`

Visibility: default

Description: `StrBufSize` returns the memory allocated for `Str`. This function ONLY gives the correct result if `Str` was allocated using `StrAlloc` (1167).

Errors: If no more memory is available, a runtime error occurs.

See also: `StrAlloc` (1167), `StrDispose` (1169)

Listing: `./sysutex/ex46.pp`

Program Example46;

{ This program demonstrates the StrBufSize function }
{ \$H+ }

Uses sysutils;

Const S = 'Some nice string';

Var P : Pchar;

Begin

 P := StrAlloc (Length(S)+1);

 StrPCopy(P,S);

 Write (P, ' has length ', length(S));

 WriteLn (' and buffer size ', StrBufSize(P));

 StrDispose(P);

End.

StrByteType

Synopsis: Return the type of byte in a null-terminated string for a multi-byte character set

Declaration: `function StrByteType(Str: PChar; Index: Cardinal) : TmbcsByteType`

Visibility: default

Description: `StrByteType` returns the type of byte in the null-terminated string `Str` at (0-based) position `Index`.

Errors: No checking on the index is performed.

See also: `TmbcsByteType` ([1085](#)), `ByteType` ([1105](#))

strcat

Synopsis: Concatenate 2 null-terminated strings.

Declaration: `function strcat(dest: pchar; source: pchar) : pchar`

Visibility: default

Description: Attaches `Source` to `Dest` and returns `Dest`.

Errors: No length checking is performed.

See also: `StrLCat` ([1172](#))

Listing: `./stringex/ex11.pp`

Program `Example11;`

Uses `strings;`

{ Program to demonstrate the StrCat function. }

Const `P1 : PChar = 'This is a PChar String.';`

Var `P2 : PChar;`

begin

`P2:= StrAlloc (StrLen(P1)*2+1);`

`StrMove (P2,P1, StrLen(P1)+1); { P2=P1 }`

`StrCat (P2,P1); { Append P2 once more }`

`Writeln ('P2 : ',P2);`

end.

strcmp

Synopsis: Compare 2 null-terminated strings, case sensitive.

Declaration: `function strcmp(str1: pchar; str2: pchar) : SizeInt`

Visibility: default

Description: Compares the null-terminated strings `S1` and `S2`. The result is

- A negative `Longint` when `S1<S2`.

- 0 when S1=S2.
- A positive Longint when S1>S2.

For an example, see StrLComp (1173).

Errors: None.

See also: StrLComp (1173), StrIComp (1171), StrLComp (1175)

strcpy

Synopsis: Copy a null-terminated string

Declaration: function strcpy(dest: pchar;source: pchar) : pchar

Visibility: default

Description: Copy the null terminated string in Source to Dest, and returns a pointer to Dest. Dest needs enough room to contain Source, i.e. StrLen(Source)+1 bytes.

Errors: No length checking is performed.

See also: StrPCopy (1178), StrLCopy (1174), StrECopy (1170)

Listing: ./stringex/ex4.pp

Program Example4;

Uses strings;

{ Program to demonstrate the StrCopy function. }

Const P : PChar = 'This is a PCHAR string.';

var PP : PChar;

begin

PP:= StrAlloc (StrLen (P)+1);

STrCopy (PP,P);

If StrComp (PP,P)<>0 **then**

 Writeln ('Oh-oh problems ... ')

else

 Writeln ('All is well : PP=',PP);

end.

StrDispose

Synopsis: Dispose of a null-terminated string on the heap.

Declaration: procedure StrDispose(Str: PChar)

Visibility: default

Description: StrDispose frees any memory allocated for Str. This function will only function correctly if Str has been allocated using StrAlloc (1167) from the SysUtils unit.

For an example, see StrBufSize (1167).

Errors: If an invalid pointer is passed, or a pointer not allocated with StrAlloc, an error may occur.

See also: StrBufSize (1167), StrAlloc (1167), StrDispose (1169)

strcpy

Synopsis: Copy a null-terminated string, return a pointer to the end.

Declaration: `function strcpy(dest: pchar;source: pchar) : pchar`

Visibility: default

Description: Copies the Null-terminated string in Source to Dest, and returns a pointer to the end (i.e. the terminating Null-character) of the copied string.

Errors: No length checking is performed.

See also: StrLCopy ([1174](#)), StrCopy ([1169](#))

Listing: ./stringex/ex6.pp

Program Example6;

Uses strings;

{ Program to demonstrate the StrECopy function. }

Const P : PChar = 'This is a PCHAR string.';

Var PP : PChar;

begin

PP:=StrAlloc (StrLen(P)+1);

If Longint(StrECopy(PP,P)) – Longint(PP) <> StrLen(P) **then**

Writeln ('Something is wrong here !')

else

Writeln ('PP= ',PP);

end.

strend

Synopsis: Return a pointer to the end of a null-terminated string

Declaration: `function strend(p: pchar) : pchar`

Visibility: default

Description: Returns a pointer to the end of P. (i.e. to the terminating null-character.

Errors: None.

See also: StrLen ([1174](#))

Listing: ./stringex/ex7.pp

Program Example6;

Uses strings;

{ Program to demonstrate the StrEnd function. }

Const P : PChar = 'This is a PCHAR string.';

```

begin
  If Longint(StrEnd(P)) - Longint(P) <> StrLen(P) then
    Writeln('Something is wrong here !')
  else
    Writeln('All is well..');
end.

```

StrFmt

Synopsis: Format a string with given arguments, store the result in a buffer.

Declaration: `function StrFmt(Buffer: PChar;Fmt: PChar;const args: Array[] of const) : Pchar`

Visibility: default

Description: `StrFmt` will format `fmt` with `Args`, as the `Format` (1141) function does, and it will store the result in `Buffer`. The function returns `Buffer`. `Buffer` should point to enough space to contain the whole result.

Errors: for a list of errors, see `Format` (1141).

See also: `StrLFmt` (1175), `FmtStr` (1140), `Format` (1141), `FormatBuf` (1146)

Listing: `./sysutex/ex80.pp`

Program `Example80`;

{ This program demonstrates the StrFmt function }

Uses `sysutils`;

Var `S` : `AnsiString`;

Begin

`SetLength(S,80)`;

`Writeln (StrFmt (@S[1], 'For some nice examples of fomatting see %s.', ['Format']))`;

End.

stricmp

Synopsis: Compare 2 null-terminated strings, case insensitive.

Declaration: `function stricmp(str1: pchar;str2: pchar) : SizeInt`

Visibility: default

Description: Compares the null-terminated strings `S1` and `S2`, ignoring case. The result is

- A negative `Longint` when `S1 < S2`.
- 0 when `S1 = S2`.
- A positive `Longint` when `S1 > S2`.

Errors: None.

See also: `StrLComp` (1173), `StrComp` (1168), `StrLComp` (1175)

Listing: ./stringex/ex8.pp

Program Example8;

Uses strings;

{ Program to demonstrate the StrLComp function. }

Const P1 : PChar = 'This is the first string.';
 P2 : PChar = 'This is the second string.';

Var L : Longint;

begin

Write ('P1 and P2 are ');
 If StrComp (P1,P2)<>0 **then write** ('NOT ');
 write ('equal. The first ');
 L:=1;
 While StrLComp(P1,P2,L)=0 **do inc** (L);
 dec(L);
 WriteLn (L, ' characters are the same.');

end.

StringReplace

Synopsis: Replace occurrences of one substring with another in a string.

Declaration: function StringReplace(const S: String;const OldPattern: String;
 const NewPattern: String;Flags: TReplaceFlags)
 : String

Visibility: default

Description: StringReplace searches the string S for occurrences of the string OldPattern and, if it is found, replaces it with NewPattern. It returns the resulting string. The behaviour of StringReplace can be runed with Flags, which is of type TReplaceFlags ([1086](#)). Standard behaviour is to replace only the first occurrence of OldPattern, and to search case sensitively.

Errors: None.

See also: TReplaceFlags ([1086](#))

strlcat

Synopsis: Concatenate 2 null-terminated strings, with length boundary.

Declaration: function strlcat(dest: pchar;source: pchar;l: SizeInt) : pchar

Visibility: default

Description: Adds MaxLen characters from Source to Dest, and adds a terminating null-character. Returns Dest.

Errors: None.

See also: StrCat ([1168](#))

Listing: ./stringex/ex12.pp

```

Program Example12;

Uses strings;

{ Program to demonstrate the StrLCat function. }

Const P1 : PChar = '1234567890';

Var P2 : PChar;

begin
  P2:=StrAlloc ( StrLen(P1)*2+1);
  P2^:=#0; { Zero length }
  StrCat (P2,P1);
  StrLCat (P2,P1,5);
  WriteLn ( 'P2 = ',P2);
end.

```

strlcomp

Synopsis: Compare limited number of characters of 2 null-terminated strings

Declaration: `function strlcomp(str1: pchar;str2: pchar;l: SizeInt) : SizeInt`

Visibility: default

Description: Compares maximum L characters of the null-terminated strings S1 and S2. The result is

- A negative Longint when S1<S2.
- 0 when S1=S2.
- A positive Longint when S1>S2.

Errors: None.

See also: StrComp ([1168](#)), StrIComp ([1171](#)), StrLComp ([1175](#))

Listing: ./stringex/ex8.pp

```

Program Example8;

Uses strings;

{ Program to demonstrate the StrLComp function. }

Const P1 : PChar = 'This is the first string.';
      P2 : PChar = 'This is the second string.';

Var L : Longint;

begin
  Write ( 'P1 and P2 are ');
  If StrComp (P1,P2)<>0 then write ( 'NOT ');
  write ( 'equal. The first ');
  L:=1;
  While StrLComp(P1,P2,L)=0 do inc (L);
  dec(L);

```

```

    WriteLn (I, ' characters are the same. ');
end.

```

strcpy

Synopsis: Copy a null-terminated string, limited in length.

Declaration: `function strcpy(dest: pchar;source: pchar;maxlen: SizeInt) : pchar`

Visibility: default

Description: Copies MaxLen characters from Source to Dest, and makes Dest a null terminated string.

Errors: No length checking is performed.

See also: StrCopy ([1169](#)), StrECopy ([1170](#))

Listing: ./stringex/ex5.pp

Program Example5;

Uses strings;

{ Program to demonstrate the StrLCopy function. }

Const P : PChar = '123456789ABCDEF';

var PP : PChar;

begin

PP:= StrAlloc(11);

WriteLn ('First 10 characters of P : ',StrLCopy (PP,P,10));

end.

strlen

Synopsis: Length of a null-terminated string.

Declaration: `function strlen(p: pchar) : sizeint`

Visibility: default

Description: Returns the length of the null-terminated string P.

Errors: None.

See also: StrNew ([1177](#))

Listing: ./stringex/ex1.pp

Program Example1;

Uses strings;

{ Program to demonstrate the StrLen function. }

Const P : PChar = 'This is a constant pchar string';

```
begin
  Writeln ( 'P          : ',p);
  Writeln ( 'length(P) : ',StrLen(P));
end.
```

StrLFmt

Synopsis: Format a string with given arguments, but with limited length.

Declaration: `function StrLFmt(Buffer: PChar;Maxlen: Cardinal;Fmt: PChar;
const args: Array[] of const) : Pchar`

Visibility: default

Description: `StrLFmt` will format `fmt` with `Args`, as the `Format` (1141) function does, and it will store maximally `Maxlen` characters of the result in `Buffer`. The function returns `Buffer`. `Buffer` should point to enough space to contain `MaxLen` characters.

Errors: for a list of errors, see `Format` (1141).

See also: `StrFmt` (1171), `FmtStr` (1140), `Format` (1141), `FormatBuf` (1146)

Listing: `./sysutex/ex81.pp`

Program Example80;

{ This program demonstrates the StrFmt function }

Uses sysutils;

Var S : AnsiString;

Begin

SetLength(S,80);

Writeln (StrLFmt (@S[1],80,'For some nice examples of fomatting see %s.',['Format']));

End.

strlicomp

Synopsis: Compare limited number of characters in 2 null-terminated strings, ignoring case.

Declaration: `function strlicomp(str1: pchar;str2: pchar;l: SizeInt) : SizeInt`

Visibility: default

Description: Compares maximum `L` characters of the null-terminated strings `S1` and `S2`, ignoring case. The result is

- A negative `Longint` when `S1<S2`.
- 0 when `S1=S2`.
- A positive `Longint` when `S1>S2`.

For an example, see `StrIComp` (1171)

Errors: None.

See also: `StrLComp` (1173), `StrComp` (1168), `StrIComp` (1171)

strlower

Synopsis: Convert null-terminated string to all-lowercase.

Declaration: `function strlower(p: pchar) : pchar`

Visibility: default

Description: Converts P to an all-lowercase string. Returns P.

Errors: None.

See also: `StrUpper` ([1186](#))

Listing: `./stringex/ex14.pp`

Program Example14;

Uses strings;

{ Program to demonstrate the StrLower and StrUpper functions. }

Const

 P1 : PChar = 'THIS IS AN UPPERCASE PCHAR STRING';

 P2 : PChar = 'this is a lowercase string';

begin

 WriteLn ('Uppercase : ', StrUpper(P2));

 StrLower (P1);

 WriteLn ('Lowercase : ', P1);

end.

strmove

Synopsis: Move a null-terminated string to new location.

Declaration: `function strmove(dest: pchar;source: pchar;l: SizeInt) : pchar`

Visibility: default

Description: Copies MaxLen characters from Source to Dest. No terminating null-character is copied. Returns Dest

Errors: None.

See also: `StrLCopy` ([1174](#)), `StrCopy` ([1169](#))

Listing: `./stringex/ex10.pp`

Program Example10;

Uses strings;

{ Program to demonstrate the StrMove function. }

Const P1 : PCHAR = 'This is a pchar string.';

Var P2 : Pchar;

```

begin
  P2:= StrAlloc (StrLen(P1)+1);
  StrMove (P2,P1,StrLen(P1)+1); { P2:=P1 }
  Writeln ('P2 = ',P2);
end.

```

strnew

Synopsis: Allocate room for new null-terminated string.

Declaration: `function strnew(p: pchar) : pchar`

Visibility: default

Description: Copies P to the Heap, and returns a pointer to the copy.

Errors: Returns Nil if no memory was available for the copy.

See also: StrCopy ([1169](#)), StrDispose ([1169](#))

Listing: ./stringex/ex16.pp

Program Example16;

Uses strings;

{ Program to demonstrate the StrNew function. }

Const P1 : PChar = 'This is a PChar string';

var P2 : PChar;

begin

 P2:=StrNew (P1);

If P1=P2 **then**

writeln ('This can't be happening...')

else

writeln ('P2 : ',P2);

end.

StrPas

Synopsis: Convert a null-terminated string to an ansistring.

Declaration: `function StrPas(Str: PChar) : String`

Visibility: default

Description: Converts a null terminated string in Str to an Ansistring, and returns this string. This string is NOT truncated at 255 characters as is the

For an example, see StrPas ([1177](#)).

Errors: None.

See also: StrPas ([1177](#))

StrPCopy

Synopsis: Copy an ansistring to a null-terminated string.

Declaration: `function StrPCopy(Dest: PChar;Source: String) : PChar`

Visibility: default

Description: `StrPCopy` Converts the Ansistring in `Source` to a Null-terminated string, and copies it to `Dest`. `Dest` needs enough room to contain the string `Source`, i.e. `Length(Source)+1` bytes.
For an example, see `StrPCopy` (1178).

Errors: No checking is performed to see whether `Dest` points to enough memory to contain `Source`.

See also: `StrPLCopy` (1178), `StrPCopy` (1178)

StrPLCopy

Synopsis: Copy a limited number of characters from an ansistring to a null-terminated string.

Declaration: `function StrPLCopy(Dest: PChar;Source: String;MaxLen: SizeUInt) : PChar`

Visibility: default

Description: `StrPLCopy` Converts maximally `MaxLen` characters of the Ansistring in `Source` to a Null-terminated string, and copies it to `Dest`. `Dest` needs enough room to contain the characters.

Errors: No checking is performed to see whether `Dest` points to enough memory to contain `L` characters of `Source`.

See also: `StrPCopy` (1178)

strpos

Synopsis: Find position of one null-terminated substring in another.

Declaration: `function strpos(str1: pchar;str2: pchar) : pchar`

Visibility: default

Description: Returns a pointer to the first occurrence of `S2` in `S1`. If `S2` does not occur in `S1`, returns `Nil`.

Errors: None.

See also: `StrScan` (1179), `StrRScan` (1179)

Listing: `./stringex/ex15.pp`

Program Example15;

Uses strings;

{ Program to demonstrate the StrPos function. }

Const P : PChar = 'This is a PChar string.';
S : PChar = 'is';

begin

WriteLn ('Position of ''is'' in P : ',longint(**StrPos**(P,S))–Longint(P));
end.

strrscan

Synopsis: Find last occurrence of a character in a null-terminated string.

Declaration: `function strrscan(p: pchar; c: Char) : pchar`

Visibility: default

Description: Returns a pointer to the last occurrence of the character C in the null-terminated string P. If C does not occur, returns Nil.

For an example, see StrScan (1179).

Errors: None.

See also: StrScan (1179), StrPos (1178)

strscan

Synopsis: Find first occurrence of a character in a null-terminated string.

Declaration: `function strscan(p: pchar; c: Char) : pchar`

Visibility: default

Description: Returns a pointer to the first occurrence of the character C in the null-terminated string P. If C does not occur, returns Nil.

Errors: None.

See also: StrRScan (1179), StrPos (1178)

Listing: ./stringex/ex13.pp

Program Example13;

Uses strings;

{ Program to demonstrate the StrScan and StrRScan functions. }

Const P : PChar = 'This is a PCHAR string.';
 S : Char = 's' ;

begin

WriteLn ('P, starting from first 's' : ', **StrScan**(P,s));

WriteLn ('P, starting from last 's' : ', **StrRScan**(P,s));

end.

StrToBool

Synopsis: Convert a string to a boolean value

Declaration: `function StrToBool(const S: String) : Boolean`

Visibility: default

Description: StrToBool will convert the string S to a boolean value. The string S can contain one of 'True', 'False' (case is ignored) or a numerical value. If it contains a numerical value, 0 is converted to False, all other values result in True. If the string S contains no valid boolean, then an EConvertError (1194) exception is raised.

Errors: On error, an `EConvertError` (1194) exception is raised.

See also: `BoolToStr` (1104)

StrToCurr

Synopsis: Convert a string to a currency value

Declaration: `function StrToCurr(const S: String) : Currency`

Visibility: default

Description: `StrToCurr` converts a string to a currency value and returns the value. The string should contain a valid currency amount, without currency symbol. If the conversion fails, an `EConvertError` (1194) exception is raised.

Errors: On error, an `EConvertError` (1194) exception is raised.

See also: `CurrToStr` (1109), `StrToCurrDef` (1180)

StrToCurrDef

Synopsis: Convert a string to a currency value, using a default value

Declaration: `function StrToCurrDef(const S: String; Default: Currency) : Currency`

Visibility: default

Description: `StrToCurrDef` converts a string to a currency value and returns the value. The string should contain a valid currency amount, without currency symbol. If the conversion fails, the fallback `Default` value is returned.

Errors: On error, the `Default` value is returned.

See also: `CurrToStr` (1109), `StrToCurr` (1180)

StrToDate

Synopsis: Convert a date string to a `TDateTime` value.

Declaration: `function StrToDate(const S: String) : TDateTime`

Visibility: default

Description: `StrToDate` converts the string `S` to a `TDateTime` date value. The Date must consist of 1 to three digits, separated by the `DateSeparator` character. If two numbers are given, they are supposed to form the day and month of the current year. If only one number is given, it is supposed to represent the day of the current month. (This is *not* supported in Delphi)

The order of the digits (y/m/d, m/d/y, d/m/y) is determined from the `ShortDateFormat` variable.

Errors: On error (e.g. an invalid date or invalid character), an `EConvertError` exception is raised.

See also: `StrToTime` (1185), `DateToStr` (1113), `TimeToStr` (1189)

Listing: `./sysutex/ex19.pp`

```

Program Example19;

{ This program demonstrates the StrToDate function }

Uses sysutils;

Procedure TestStr (S : String);

begin
  WriteIn (S, ' : ', DateToStr(StrToDate(S)));
end;

Begin

  WriteIn ( 'ShortDateFormat ', ShortDateFormat);
  TestStr(DateTimeToStr(Date));
  TestStr( '05/05/1999 ');
  TestStr( '5/5 ');
  TestStr( '5 ');
End.

```

StrToDateTime

Synopsis: Convert a date/time string to a TDateTime value.

Declaration: function StrToDateTime(const S: String) : TDateTime

Visibility: default

Description: StrToDateTime converts the string S to a TDateTime date and time value. The Date must consist of 1 to three digits, separated by the DateSeparator character. If two numbers are given, they are supposed to form the day and month of the current year. If only one number is given, it is supposed to represent the day of the current month. (This is *not* supported in Delphi)

The order of the digits (y/m/d, m/d/y, d/m/y) is determined from the ShortDateFormat variable.

Errors: On error (e.g. an invalid date or invalid character), an EConvertError exception is raised.

See also: StrToDate ([1180](#)), StrToTime ([1185](#)), DateTimeToStr ([1111](#))

Listing: ./sysutex/ex20.pp

```

Program Example20;

{ This program demonstrates the StrToDateTime function }

Uses sysutils;

Procedure TestStr (S : String);

begin
  WriteIn (S, ' : ', DateTimeToStr(StrToDateTime(S)));
end;

Begin

  WriteIn ( 'ShortDateFormat ', ShortDateFormat);

```

```

TestStr(DateTimeToStr(Now));
TestStr('05-05-1999 15:50');
TestStr('5-5 13:30');
TestStr('5 1:30PM');
End.

```

StrToFloat

Synopsis: Convert a string to a floating-point value.

Declaration: `function StrToFloat(const S: String) : Extended`

Visibility: default

Description: `StrToFloat` converts the string `S` to a floating point value. `S` should contain a valid stroing representation of a floating point value (either in decimal or scientific notation). If the string contains a decimal value, then the decimal separator character can either be a `'.'` or the value of the `DecimalSeparator` variable.

Errors: If the string `S` doesn't contain a valid floating point string, then an exception will be raised.

See also: `TextToFloat` ([1187](#)), `FloatToStr` ([1137](#)), `FormatFloat` ([1147](#)), `StrToInt` ([1183](#))

Listing: `./sysutex/ex90.pp`

Program Example90;

```

{ This program demonstrates the StrToFloat function }
{$mode objfpc}
{$h+ }

```

Uses SysUtils;

Const

```

NrValues = 5;
TestStr : Array[1..NrValues] of string =
    ('1,1', '-0,2', '1,2E-4', '0', '1E4');

```

Procedure Testit;

Var

```

I : Integer;
E : Extended;

```

begin

```

WriteLn('Using DecimalSeparator : ', DecimalSeparator);
For I:=1 to NrValues do
begin
    WriteLn('Converting : ', TestStr[I]);
    Try
        E:=StrToFloat(TestStr[I]);
        WriteLn('Converted value : ', E);
    except
        On E : Exception do
            WriteLn('Exception when converting : ', E.Message);
    end;
end;
end;

```

```

Begin
  DecimalSeparator:= ',';
  Testit;
  DecimalSeparator:= '.';
  Testit;
End.

```

StrToFloatDef

Synopsis: Convert a string to a float, with a default value.

Declaration: `function StrToFloatDef(const S: String;const Default: Extended) : Extended`

Visibility: default

Description: `StrToFloatDef` tries to convert the string `S` to a floating point value, and returns this value. If the conversion fails for some reason, the value `Default` is returned instead.

Errors: None. On error, the `Default` value is returned.

StrToInt

Synopsis: Convert a string to an integer value.

Declaration: `function StrToInt(const s: String) : Integer`

Visibility: default

Description: `StrToInt` will convert the string `Sto` to an integer. If the string contains invalid characters or has an invalid format, then an `EConvertError` is raised.

To be successfully converted, a string can contain a combination of numerical characters, possibly preceded by a minus sign (-). Spaces are not allowed.

Errors: In case of error, an `EConvertError` is raised.

See also: `IntToStr` ([1157](#)), `StrToIntDef` ([1184](#))

Listing: `./sysutex/ex82.pp`

Program `Example82;`

```
{ $mode objfpc }
```

```
{ This program demonstrates the StrToInt function }
```

```
Uses sysutils;
```

```
Begin
```

```
  Writeln ( StrToInt( '1234' ));
```

```
  Writeln ( StrToInt( '-1234' ));
```

```
  Writeln ( StrToInt( '0' ));
```

```
  Try
```

```
    Writeln ( StrToInt( '12345678901234567890' ));
```

```
  except
```

```
    On E : EConvertError do
```

```

    WriteLn ('Invalid number encountered');
end;
End.

```

StrToInt64

Synopsis: Convert a string to an Int64 value.

Declaration: `function StrToInt64(const s: String) : Int64`

Visibility: default

Description: `StrToInt64` converts the string `S` to a `Int64` value, and returns this value. The string can only contain numerical characters, and optionally a minus sign as the first character. Whitespace is not allowed.

Hexadecimal values (starting with the `$` character) are supported.

Errors: On error, a `EConvertError` ([1194](#)) exception is raised.

See also: `TryStrToInt64` ([1193](#)), `StrToInt64Def` ([1184](#)), `StrToInt` ([1183](#)), `TryStrToInt` ([1192](#)), `StrToIntDef` ([1184](#))

StrToInt64Def

Synopsis: Convert a string to an Int64 value, with a default value

Declaration: `function StrToInt64Def(const S: String;Default: Int64) : Int64`

Visibility: default

Description: `StrToInt64Def` tries to convert the string `S` to a `Int64` value, and returns this value. If the conversion fails for some reason, the value `Default` is returned instead.

Errors: None. On error, the `Default` value is returned.

See also: `StrToInt64` ([1184](#)), `TryStrToInt64` ([1193](#)), `StrToInt` ([1183](#)), `TryStrToInt` ([1192](#)), `StrToIntDef` ([1184](#))

StrToIntDef

Synopsis: Convert a string to an integer value, with a default value.

Declaration: `function StrToIntDef(const S: String;Default: Integer) : Integer`

Visibility: default

Description: `StrToIntDef` will convert a string to an integer. If the string contains invalid characters or has an invalid format, then `Default` is returned.

To be successfully converted, a string can contain a combination of numerical characters, possibly preceded by a minus sign (`-`). Spaces are not allowed.

Errors: None.

See also: `IntToStr` ([1157](#)), `StrToInt` ([1183](#))

Listing: `./sysutex/ex83.pp`

```

Program Example82;

{$mode objfpc}

{ This program demonstrates the StrToInt function }

Uses sysutils;

Begin
  WriteLn ( StrToIntDef( '1234' ,0));
  WriteLn ( StrToIntDef( '-1234' ,0));
  WriteLn ( StrToIntDef( '0' ,0));
  Try
    WriteLn ( StrToIntDef( '12345678901234567890' ,0));
  except
    On E : EConvertError do
      WriteLn ( 'Invalid number encountered' );
  end;
End.

```

StrToTime

Synopsis: Convert a time string to a `TDateTime` value.

Declaration: `function StrToTime(const S: String) : TDateTime`

Visibility: default

Description: `StrToTime` converts the string `S` to a `TDateTime` time value. The time must consist of 1 to 4 digits, separated by the `TimeSeparator` character. If two numbers are given, they are supposed to form the hour and minutes.

Errors: On error (e.g. an invalid date or invalid character), an `EConvertError` exception is raised.

See also: `StrToDate` ([1180](#)), `StrToDateTime` ([1181](#)), `TimeToStr` ([1189](#))

Listing: `./sysutex/ex21.pp`

```

Program Example21;

{ This program demonstrates the StrToTime function }

Uses sysutils;

Procedure TestStr (S : String);

begin
  WriteLn (S, ' : ', TimeToStr(StrToTime(S)));
end;

Begin
  teststr ( TimeToStr(Time));
  teststr ( '12:00' );
  teststr ( '15:30' );
  teststr ( '3:30PM' );
End.

```

strupper

Synopsis: Convert null-terminated string to all-uppercase

Declaration: `function strupper(p: pchar) : pchar`

Visibility: default

Description: Converts P to an all-uppercase string. Returns P.

For an example, see `StrLower` ([1176](#))

Errors: None.

See also: `StrLower` ([1176](#))

SysErrorMessage

Synopsis: Format a system error message.

Declaration: `function SysErrorMessage(ErrorCode: Integer) : String`

Visibility: default

Description: `SysErrorMessage` returns a string that describes the operating system error code `ErrorCode`.

Errors: This routine may not be implemented on all platforms.

See also: `EOSError` ([1197](#))

SystemTimeToDateTime

Synopsis: Convert a system time to a `TDateTime` value.

Declaration: `function SystemTimeToDateTime(const SystemTime: TSystemTime) : TDateTime`

Visibility: default

Description: `SystemTimeToDateTime` converts a `TSystemTime` record to a `TDateTime` style date/time indication.

Errors: None.

See also: `DateTimeToSystemTime` ([1112](#))

Listing: `./sysutex/ex22.pp`

Program `Example22`;

{ This program demonstrates the SystemTimeToDateTime function }

Uses `sysutils`;

Var `ST : TSystemTime`;

Begin

`DateTimeToSystemTime(Now, ST);`

With `St` **do**

begin

`Writeln ('Today is ', year, '/', month, '/', Day);`

`Writeln ('The time is ', Hour, ':', minute, ':', Second, '.', MilliSecond);`

```

    end;
    Writeln ( 'Converted : ', DateTimeToStr(SystemTimeToDateTime(ST)));
End.

```

TextToFloat

Synopsis: Convert a buffer to a float value.

Declaration: `function TextToFloat(Buffer: PChar; var Value: Extended) : Boolean`
`function TextToFloat(Buffer: PChar; var Value; ValueType: TFloatValue)`
`: Boolean`

Visibility: default

Description: `TextToFloat` converts the string in `Buffer` to a floating point value. `Buffer` should contain a valid stroing representation of a floating point value (either in decimal or scientific notation). If the buffer contains a decimal value, then the decimal separator character can either be a '.' or the value of the `DecimalSeparator` variable.

The function returns `True` if the conversion was successful.

Errors: If there is an invalid character in the buffer, then the function returns `False`

See also: `StrToFloat` ([1182](#)), `FloatToStr` ([1137](#)), `FormatFloat` ([1147](#))

Listing: ./sysutex/ex91.pp

Program Example91;

```

{ This program demonstrates the TextToFloat function }
{$mode objfpc}
{$h+ }

```

Uses SysUtils;

Const

```

    NrValues = 5;
    TestStr : Array[1..NrValues] of pchar =
        ('1,1 ', '-0,2 ', '1,2E-4 ', '0 ', '1E4 ');

```

Procedure Testit;

Var

```

    I : Integer;
    E : Extended;

```

begin

```

    Writeln('Using DecimalSeparator : ', DecimalSeparator);
    For I:=1 to NrValues do
        begin
            Writeln('Converting : ', TestStr[I]);
            If TextToFloat(TestStr[I], E) then
                Writeln('Converted value : ', E)
            else
                Writeln('Unable to convert value. ');
        end;

```

end;

```

Begin
  DecimalSeparator:= ',';
  Testit;
  DecimalSeparator:= '.';
  Testit;
End.

```

Time

Synopsis: Returns the current time.

Declaration: function Time : TDateTime

Visibility: default

Description: Time returns the current time in TDateTime format. The date part of the TDateTimeValue is set to zero.

Errors: None.

See also: Now ([1162](#)), Date ([1110](#))

Listing: ./sysutex/ex23.pp

Program Example23;

{ This program demonstrates the Time function }

Uses sysutils;

```

Begin
  WriteLn ( 'The time is : ',TimeToStr(Time));
End.

```

TimeStampToDateTime

Synopsis: Convert a TimeStamp value to a TDateTime value.

Declaration: function TimeStampToDateTime(const TimeStamp: TTimeStamp) : TDateTime

Visibility: default

Description: TimeStampToDateTime converts TimeStamp to a TDateTime format variable. It is the inverse operation of DateTimeToTimeStamp ([1113](#)).

Errors: None.

See also: DateTimeToTimeStamp ([1113](#)), TimeStampToMsecs ([1189](#))

Listing: ./sysutex/ex24.pp

Program Example24;

{ This program demonstrates the TimeStampToDateTime function }

Uses sysutils;

```

Var TS : TTimeStamp;
      DT : TDateTime;

Begin
  TS:=DateTimeToTimeStamp (Now);
  With TS do
    begin
      WriteLn ( 'Now is ',time, ' millisecond past midnight');
      WriteLn ( 'Today is ',Date, ' days past 1/1/0001');
    end;
  DT:=TimeStampToDateTime(TS);
  WriteLn ( 'Together this is : ',DateTimeToStr(DT));
End.

```

TimeStampToMSecs

Synopsis: Converts a timestamp to a number of milliseconds.

Declaration: function TimeStampToMSecs(const TimeStamp: TTimeStamp) : comp

Visibility: default

Description: TimeStampToMSecs converts TimeStamp to the number of seconds since 1/1/0001.

Use TTimeStamp variables if you need to keep very precise track of time.

For an example, see MSecsToTimeStamp ([1161](#)).

Errors: None.

See also: MSecsToTimeStamp ([1161](#)), TimeStampToDateTime ([1188](#))

TimeToStr

Synopsis: Convert a TDateTime time to a string using a predefined format.

Declaration: function TimeToStr(Time: TDateTime) : String

Visibility: default

Description: TimeToStr converts the time in Time to a string. It uses the ShortTimeFormat variable to see what formatting needs to be applied. It is therefor entirely equivalent to a FormatDateTime('t', Time) call.

Errors: None.

Listing: ./sysutex/ex25.pp

Program Example25;

{ This program demonstrates the TimeToStr function }

Uses sysutils;

Begin

```

  WriteLn ( 'The current time is : ',TimeToStr(Time));
End.

```

Trim

Synopsis: Trim whitespace from the ends of a string.

Declaration: `function Trim(const S: String) : String`

Visibility: default

Description: Trim strips blank characters (spaces) at the beginning and end of S and returns the resulting string.

Only #32 characters are stripped.

If the string contains only spaces, an empty string is returned.

Errors: None.

See also: TrimLeft ([1190](#)), TrimRight ([1191](#))

Listing: ./sysutex/ex84.pp

Program Example84;

{ This program demonstrates the Trim function }

Uses sysutils;

{ \$H+ }

Procedure Testit (S : **String**);

begin

WriteLn (' ', Trim(S), ' ');

end;

Begin

 Testit (' ha ha what gets lost ? ');

 Testit (#10#13'haha ');

 Testit (' ');

End.

TrimLeft

Synopsis: Trim whitespace from the beginning of a string.

Declaration: `function TrimLeft(const S: String) : String`

Visibility: default

Description: TrimLeft strips blank characters (spaces) at the beginning of S and returns the resulting string.

Only #32 characters are stripped. If the string contains only spaces, an empty string is returned.

Errors: None.

See also: Trim ([1190](#)), TrimRight ([1191](#))

Listing: ./sysutex/ex85.pp

Program Example85;

{ This program demonstrates the TrimLeft function }

Uses sysutils;

```

{$H+}

Procedure Testit (S : String);

begin
    WriteIn ( ' ', TrimLeft(S), ' ');
end;

Begin
    Testit ( '  ha ha what gets lost ? ');
    Testit (#10#13'haha ');
    Testit ( '          ');
End.

```

TrimRight

Synopsis: Trim whitespace from the end of a string.

Declaration: `function TrimRight(const S: String) : String`

Visibility: default

Description: Trim strips blank characters (spaces) at the end of S and returns the resulting string. Only #32 characters are stripped. If the string contains only spaces, an empty string is returned.

Errors: None.

See also: Trim ([1190](#)), TrimLeft ([1190](#))

Listing: ./sysutex/ex86.pp

Program Example86;

{ This program demonstrates the TrimRight function }

Uses sysutils;
{\$H+}

Procedure Testit (S : **String**);

begin
 WriteIn (' ', TrimRight(S), ' ');
end;

Begin
 Testit (' ha ha what gets lost ? ');
 Testit (#10#13'haha ');
 Testit (' ');
End.

TryEncodeDate

Synopsis: Try to encode a date, and indicate success.

Declaration: `function TryEncodeDate(Year: Word;Month: Word;Day: Word;
 var Date: TDateTime) : Boolean`

Visibility: default

Description: `TryEncodeDate` will check the validity of the `Year`, `Month` and `Day` arguments, and if they are all valid, then they will be encoded as a `TDateTime` value and returned in `D`. The function will return `True` in this case. If an invalid argument is passed, then `False` will be returned.

Errors: None. If an error occurs during the encoding, `False` is returned.

See also: `EncodeDate` (1119), `DecodeDateFully` (1115), `DecodeDate` (1114), `TryEncodeTime` (1192)

TryEncodeTime

Synopsis: Try to encode a time, and indicate success.

Declaration: `function TryEncodeTime(Hour: Word;Min: Word;Sec: Word;MSec: Word;
var Time: TDateTime) : Boolean`

Visibility: default

Description: `TryEncodeTime` will check the validity of the `Hour`, `Min`, `Sec` and `MSec` arguments, and will encode them in a `TDateTime` value which is returned in `T`. If the arguments are valid, then `True` is returned, otherwise `False` is returned.

Errors: None. If an error occurs during the encoding, `False` is returned.

See also: `EncodeTime` (1119), `DecodeTime` (1115), `TryEncodeDate` (1191)

TryFloatToCurr

Synopsis: Try to convert a float value to a currency value and report on success.

Declaration: `function TryFloatToCurr(const Value: Extended;var AResult: Currency)
: Boolean`

Visibility: default

Description: `TryFloatToCurr` tries convert the `Value` floating point value to a `Currency` value. If successful, the function returns `True` and the resulting currency value is returned in `AResult`. It checks whether `Value` is in the valid range of currencies (determined by `MinCurrency` (1080) and `MaxCurrency` (1080)). If not, `False` is returned.

Errors: If `Value` is out of range, `False` is returned.

See also: `FloatToCurr` (1136), `MinCurrency` (1080), `MaxCurrency` (1080)

TryStrToInt

Synopsis: Try to convert a string to an integer, and report on success.

Declaration: `function TryStrToInt(const s: String;var i: Integer) : Boolean`

Visibility: default

Description: `TryStrToInt` tries to convert the string `S` to an integer, and returns `True` if this was successful. In that case the converted integer is returned in `I`. If the conversion failed, (an invalid string, or the value is out of range) then `False` is returned.

Errors: None. On error, `False` is returned.

See also: `StrToInt` (1183), `TryStrToInt64` (1193), `StrToIntDef` (1184), `StrToInt64` (1184), `StrToInt64Def` (1184)

TryStrToInt64

Synopsis: Try to convert a string to an int64 value, and report on success.

Declaration: `function TryStrToInt64(const s: String; var i: Int64) : Boolean`

Visibility: default

Description: `TryStrToInt64` tries to convert the string `S` to a `Int64` value, and returns this value in `I` if successful. If the conversion was successful, the function result is `True`, or `False` otherwise. The string can only contain numerical characters, and optionally a minus sign as the first character. Whitespace is not allowed.

Hexadecimal values (starting with the `$` character) are supported.

Errors: None. On error, `False` is returned.

See also: `StrToInt64` ([1184](#)), `StrToInt64Def` ([1184](#)), `StrToInt` ([1183](#)), `TryStrToInt` ([1192](#)), `StrToIntDef` ([1184](#))

UpperCase

Synopsis: Return an uppercase version of a string.

Declaration: `function UpperCase(const s: String) : String`

Visibility: default

Description: `UpperCase` returns the uppercase equivalent of `S`. Ansi characters are not taken into account, only ASCII codes below 127 are converted. It is completely equivalent to the `UpCase` function of the system unit, and is provided for compatibility only.

Errors: None.

See also: `AnsiLowerCase` ([1093](#)), `LowerCase` ([1160](#)), `AnsiUpperCase` ([1101](#))

Listing: `./sysutex/ex87.pp`

Program `Example87;`

{ This program demonstrates the UpperCase function }

Uses `sysutils;`

Begin

`WriteLn (UpperCase('this will come OUT ALL uPpErCaSe !'));`
End.

WrapText

Declaration: `function WrapText(const Line: String; const BreakStr: String;
const BreakChars: TSysCharSet; MaxCol: Integer) : String`
`function WrapText(const Line: String; MaxCol: Integer) : String`

Visibility: default

30.13 EAbort

Description

`Abort` is raised by the `Abort` (1088) procedure. It is not displayed in GUI applications, and serves only to immediately abort the current procedure, and return control to the main program loop.

30.14 EAbstractError

Description

`EAbstractError` is raised when an abstract error occurs, i.e. when an unimplemented abstract method is called.

30.15 EAccessViolation

Description

`EAccessViolation` is raised when the OS reports an Access Violation, i.e. when invalid memory is accessed.

30.16 EAssertionFailed

Description

`EAssertionFailed` is raised when an application that is compiled with assertions, encounters an invalid assertion.

30.17 EControlC

Description

`EControlC` is raised when the user has pressed CTRL-C in a console application.

30.18 EConvertError

Description

`EConvertError` is raised by the various conversion routines in the `SysUtils` unit. The message will contain more specific error information.

30.19 EDivByZero

Description

`EDivByZero` is used when the operating system or CPU signals a division by zero error.

30.20 EExternal

Description

EExternal is the base exception for all external exceptions, as reported by the CPU or operating system, as opposed to internal exceptions, which are raised by the program itself. The SysUtils unit converts all operating system errors to descendents of EExternal.

30.21 EExternalException

Description

EExternalException is raised when an external routine raises an exception.

30.22 EHeapMemoryError

Description

EHeapMemoryError is raised when an error occurs in heap (dynamically allocated) memory.

Method overview

Page	Method	Description
1195	FreeInstance	

EHeapMemoryError.FreeInstance

Declaration: `procedure FreeInstance; Override`

Visibility: `protected`

30.23 EInOutError

Description

EInOutError is raised when a IO routine of Free Pascal returns an error. The error is converted to an EInOutError only if the input/output checking feature of FPC is turned on. The error code of the input/output operation is returned in ErrorCode (??).

30.24 EInterror

Description

EInterror is used when the operating system or CPU signals an integer operation error, e.g., an overflow.

30.25 EIntfCastError

Description

`EIntfCastError` is raised when an invalid interface cast is encountered.

30.26 EIntOverflow

Description

`EIntOverflow` is used when the operating system or CPU signals a integer overflow error.

30.27 EInvalidCast

Description

`EInvalidCast` is raised when an invalid typecast error (using the `as` operator) is encountered.

30.28 EInvalidContainer

Description

`EInvalidContainer` is not yet used by Free Pascal, and is provided for Delphi compatibility only.

30.29 EInvalidInsert

Description

`EInvalidInsert` is not yet used by Free Pascal, and is provided for Delphi compatibility only.

30.30 EInvalidOp

Description

`EInvalidOp` is raised when an invalid operation is encountered.

30.31 EInvalidPointer

Description

`EInvalidPointer` is raised when an invalid heap pointer is used.

30.32 EMathError

Description

EMathError is used when the operating system or CPU signals a floating point overflow error.

30.33 ENoThreadSupport

Description

ENoThreadSupport is raised when some thread routines are invoked, and thread support was not enabled when the program was compiled.

30.34 EOSError

Description

EOSError is raised when some Operating System call fails. The ErrorCode (??) property contains the operating system error code.

30.35 EOutOfMemory

Description

EOutOfMemory occurs when memory can no longer be allocated on the heap. An instance of EOutOfMemory is allocated on the heap at program startup, so it is available when needed.

30.36 EOverflow

Description

EOverflow occurs when a float operation overflows. (i.e. result is too big to represent).

30.37 EPackageError

Description

EPackageError is not yet used by Free Pascal, and is provided for Delphi compatibility only.

30.38 EPrivilege

Description

EPrivilege is raised when the OS reports that an invalid instruction was executed.

30.39 EPropReadOnly

Description

EPropReadOnly is raised when an attempt is made to write to a read-only property.

30.40 EPropWriteOnly

Description

EPropWriteOnly is raised when an attempt is made to read from a write-only property.

30.41 ERangeError

Description

ERangeError is raised by the Free Pascal runtime library if range checking is on, and a range check error occurs.

30.42 ESafecallException

Description

ESafecallException is not yet used by Free Pascal, and is provided for Delphi compatibility only.

30.43 EStackOverflow

Description

EStackOverflow occurs when the stack has grown too big (e.g. by infinite recursion).

30.44 EUnderflow

Description

EOverflow occurs when a float operation underflows (i.e. result is too small to represent).

30.45 EVariantError

Description

EVariantError is raised by the internal variant routines.

30.46 Exception

Description

`Exception` is the base class for all exception handling routines in the RTL and FCL. While it is possible to raise an exception with any class descending from `TObject`, it is recommended to use `Exception` as the basis of exception class objects: the `Exception` class introduces properties to associate a message and a help context with the exception being raised. What is more, the `SysUtils` unit sets the necessary hooks to catch and display unhandled exceptions: in such cases, the message displayed to the end user, will be the message stored in the exception class.

Method overview

Page	Method	Description
1199	<code>Create</code>	Constructs a new exception object with a given message.
1199	<code>CreateFmt</code>	Constructs a new exception object and formats a new message.
1200	<code>CreateFmtHelp</code>	Constructs a new exception object and sets the help context and formats the message
1200	<code>CreateHelp</code>	Constructs a new exception object and sets the help context.
1200	<code>CreateRes</code>	Constructs a new exception object and gets the message from a resource.
1200	<code>CreateResFmt</code>	Constructs a new exception object and formats the message from a resource.
1201	<code>CreateResFmtHelp</code>	Constructs a new exception object and sets the help context and formats the message from a resource
1201	<code>CreateResHelp</code>	Constructs a new exception object and sets the help context and gets the message from a resource

Property overview

Page	Property	Access	Description
1201	<code>HelpContext</code>	rw	Help context associated with the exception.
1201	<code>Message</code>	rw	Message associated with the exception.

Exception.Create

Synopsis: Constructs a new exception object with a given message.

Declaration: `constructor Create(const msg: String)`

Visibility: `public`

Errors: Construction may fail if there is not enough memory on the heap.

See also: `Exception.CreateFmt` ([1199](#)), `Exception.Message` ([1201](#))

Exception.CreateFmt

Synopsis: Constructs a new exception object and formats a new message.

Declaration: `constructor CreateFmt(const msg: String; const args: Array[] of const)`

Visibility: `public`

Errors: Construction may fail if there is not enough memory on the heap.

See also: [Exception.Create \(1199\)](#), [Exception.Message \(1201\)](#), [Format \(1141\)](#)

Exception.CreateRes

Synopsis: Constructs a new exception object and gets the message from a resource.

Declaration: `constructor CreateRes(ResString: PString)`

Visibility: `public`

Errors: Construction may fail if there is not enough memory on the heap.

See also: [Exception.Create \(1199\)](#), [Exception.CreateFmt \(1199\)](#), [Exception.CreateResFmt \(1200\)](#), [Exception.Message \(1201\)](#)

Exception.CreateResFmt

Synopsis: Constructs a new exception object and formats the message from a resource.

Declaration: `constructor CreateResFmt(ResString: PString;
const Args: Array[] of const)`

Visibility: `public`

Description: `CreateResFmt` does the same as `CreateFmt (1199)`, but fetches the message from the resource string `ResString`.

Errors: Construction may fail if there is not enough memory on the heap.

See also: [Exception.Create \(1199\)](#), [Exception.CreateFmt \(1199\)](#), [Exception.CreateRes \(1200\)](#), [Exception.Message \(1201\)](#)

Exception.CreateHelp

Synopsis: Constructs a new exception object and sets the help context.

Declaration: `constructor CreateHelp(const Msg: String; AHelpContext: Integer)`

Visibility: `public`

Description: `CreateHelp` does the same as the `Create (1199)` constructor, but additionally stores `AHelpContext` in the `HelpContext (1201)` property.

See also: [Exception.Create \(1199\)](#)

Exception.CreateFmtHelp

Synopsis: Constructs a new exception object and sets the help context and formats the message

Declaration: `constructor CreateFmtHelp(const Msg: String;
const Args: Array[] of const;
AHelpContext: Integer)`

Visibility: `public`

Description: `CreateFmtHelp` does the same as the `CreateFmt (1199)` constructor, but additionally stores `AHelpContext` in the `HelpContext (1201)` property.

See also: [Exception.CreateFmt \(1199\)](#)

Exception.CreateResHelp

Synopsis: Constructs a new exception object and sets the help context and gets the message from a resource

Declaration: `constructor CreateResHelp(ResString: PString; AHelpContext: Integer)`

Visibility: `public`

Description: `CreateResHelp` does the same as the `CreateRes` (1200) constructor, but additionally stores `AHelpContext` in the `HelpContext` (1201) property.

See also: `Exception.CreateRes` (1200)

Exception.CreateResFmtHelp

Synopsis: Constructs a new exception object and sets the help context and formats the message from a resource

Declaration: `constructor CreateResFmtHelp(ResString: PString;
const Args: Array[] of const;
AHelpContext: Integer)`

Visibility: `public`

Description: `CreateResFmtHelp` does the same as the `CreateResFmt` (1200) constructor, but additionally stores `AHelpContext` in the `HelpContext` (1201) property.

See also: `Exception.CreateResFmt` (1200)

Exception.HelpContext

Synopsis: Help context associated with the exception.

Declaration: `Property HelpContext : LongInt`

Visibility: `public`

Access: `Read, Write`

Description: `HelpContext` is the help context associated with the exception, and can be used to provide context-sensitive help when the exception error message is displayed. It should be set in the exception constructor.

See also: `Exception.CreateHelp` (1200), `Exception.Message` (1201)

Exception.Message

Synopsis: Message associated with the exception.

Declaration: `Property Message : String`

Visibility: `public`

Access: `Read, Write`

Description: `Message` provides additional information about the exception. It is shown to the user in e.g. the `ShowException` (1166) routine, and should be set in the constructor when the exception is raised.

See also: `Exception.Create` (1199), `Exception.HelpContext` (1201)

30.47 EZeroDivide

Description

EZeroDivide occurs when a float division by zero occurs.

Chapter 31

Reference for unit 'typinfo'

31.1 Auxiliary functions

Other typinfo related functions.

Table 31.1:

Name	Description
GetEnumName (1210)	Get an enumerated type element name
GetEnumValue (1212)	Get ordinal number of an enumerated tye, based on the name.
GetTypeData (1223)	Skip type name and return a pointer to the type data
SetToString (1231)	Convert a set to its string representation
StringToSet (1232)	Convert a string representation of a set to a set

31.2 Getting or setting property values

Functions to set or set a property's value.

31.3 Examining published property information

Functions for retrieving or examining property information

31.4 Used units

31.5 Overview

The `TypeInfo` unit contains many routines which can be used for the querying of the Run-Time Type Information (RTTI) which is generated by the compiler for classes that are compiled under the `{ $M+ }` switch. This information can be used to retrieve or set property values for published properties for totally unknown classes. In particular, it can be used to stream classes. The `TPersistent` class in the `Classes` unit is compiled in the `{ $M+ }` state and serves as the base class for all classes that need to be streamed.

Table 31.2:

Name	Description
<code>GetEnumProp</code> (1211)	Return the value of an enumerated type property
<code>GetFloatProp</code> (1212)	Return the value of a float property
<code>GetInt64Prop</code> (1213)	Return the value of an Int64 property
<code>GetMethodProp</code> (1214)	Return the value of a procedural type property
<code>GetObjectProp</code> (1216)	Return the value of an object property
<code>GetOrdProp</code> (1217)	Return the value of an ordinal type property
<code>GetProperty</code> (1221)	Return the value of a property as a variant
<code>GetSetProp</code> (1221)	Return the value of a set property
<code>GetStrProp</code> (1223)	Return the value of a string property
<code>GetVariantProp</code> (1224)	Return the value of a variant property
<code>SetEnumProp</code> (1227)	Set the value of an enumerated type property
<code>SetFloatProp</code> (1228)	Set the value of a float property
<code>SetInt64Prop</code> (1228)	Set the value of an Int64 property
<code>SetMethodProp</code> (1228)	Set the value of a procedural type property
<code>SetObjectProp</code> (1229)	Set the value of an object property
<code>SetOrdProp</code> (1229)	Set the value of an ordinal type property
<code>SetPropertyValue</code> (1230)	Set the value of a property through a variant
<code>SetSetProp</code> (1230)	Set the value of a set property
<code>SetStrProp</code> (1231)	Set the value of a string property
<code>SetVariantProp</code> (1232)	Set the value of a variant property

The unit should be compatible to the Delphi 5 unit with the same name. The only calls that are still missing are the Variant calls, since Free Pascal does not support the variant type yet.

The examples in this chapter use a `rttiobj` file, which contains an object that has a published property of all supported types. It also contains some auxiliary routines and definitions.

31.6 Constants, types and variables

Constants

```
BooleanIdents : Array[Boolean] of String = ('False', 'True' )
```

Names for boolean values

```
DotSep : String = '.'
```

Name separator character

```
ptConst = 3
```

Constant used in acces method

```
ptField = 0
```

Property acces directly from field

```
ptStatic = 1
```

Table 31.3:

Name	Description
FindPropInfo (1208)	Getting property type information, With error checking.
GetPropInfo (1218)	Getting property type information, No error checking.
GetPropInfos (1219)	Find property information of a certain kind
GetObjectPropClass (1217)	Return the declared class of an object property
GetPropList (1220)	Get a list of all published properties
IsPublishedProp (1224)	Is a property published
IsStoredProp (1225)	Is a property stored
PropIsType (1226)	Is a property of a certain kind
PropType (1226)	Return the type of a property

Table 31.4: Used units by unit 'typinfo'

Name	Page
sysutils	1071

Property acces via static method

```
ptVirtual = 2
```

Property acces via virtual method

```
tkAny = [Low ( TTypeKind ) ..High ( TTypeKind ) ]
```

Any property type

```
tkMethods = [tkMethod]
```

Only method properties. (event handlers)

```
tkProperties = tkAny - tkMethods - [tkUnknown]
```

Real properties. (not methods)

```
tkString = tkSString
```

Alias for the tsSString enumeration value

Types

```
PPropInfo = ^TPropInfo
```

Pointer to TPropInfo ([1207](#)) record

```
PPropList = ^TPropList
```

Pointer to TPropList ([1207](#))

```
PTypeInfo = ^PTypeInfo
```

Pointer to PTypeInfo (1206) pointer

PTypeData = ^TTypeData

Pointer to TTypeData (1208) record.

PTypeInfo = ^TTypeInfo

Pointer to TTypeInfo (1208) record

TFloatType = (ftSingle,ftDouble,ftExtended,ftComp,ftCurr)

Table 31.5: Enumeration values for type TFloatType

Value	Explanation
ftComp	Comp-type float
ftCurr	Currency-type float
ftDouble	Double-sized float
ftExtended	Extended-size float
ftSingle	Single-sized float

The size of a float type.

TIntfFlag = (ifHasGuid,ifDispInterface,ifDispatch)

Table 31.6: Enumeration values for type TIntfFlag

Value	Explanation
ifDispatch	Interface is a dispatch interface
ifDispInterface	Interface is a dual dispatch interface
ifHasGuid	Interface has GUID identifier

Type of interface.

TIntfFlags= Set of (ifDispatch,ifDispInterface,ifHasGuid)

Set of TIntfFlag (1206).

TIntfFlagsBase= Set of (ifDispatch,ifDispInterface,ifHasGuid)

Set of TIntfFlag (1206).

TMethodKind = (mkProcedure,mkFunction,mkConstructor,mkDestructor,
mkClassProcedure,mkClassFunction)

Method type description

TParamFlags= Set of (pfVar,pfConst,pfArray,pfAddress,pfReference,pfOut)

The kind of parameter for a method

Table 31.7: Enumeration values for type TMethodKind

Value	Explanation
mkClassFunction	Class function
mkClassProcedure	Class procedure
mkConstructor	Class constructor
mkDestructor	Class Desctructor
mkFunction	Function method
mkProcedure	Procedure method.

TProcInfoProc = procedure(PropInfo: PPropInfo) of object

Property info callback method

```
TPropData = packed record
  PropCount : Word;
  PropList : record
    _alignmentdummy : puint;
  end;
end
```

The TPropData record is not used, but is provided for completeness and compatibility with Delphi.

```
TPropInfo = packed record
  PropType : PTypeInfo;
  GetProc : Pointer;
  SetProc : Pointer;
  StoredProc : Pointer;
  Index : Integer;
  Default : LongInt;
  NameIndex : SmallInt;
  PropProcs : Byte;
  Name : ShortString;
end
```

The TPropInfo record describes one published property of a class. The property information of a class are stored as an array of TPropInfo records.

The Name field is stored not with 255 characters, but with just as many characters as required to store the name.

TPropList = Array[0..65535] of PPropInfo

Array of property information pointers

TOrdType = (otSByte, otUByte, otSWord, otUWord, otSLong, otULong)

If the property is and ordinal type, then TOrdType determines the size and sign of the ordinal type:

Table 31.8: Enumeration values for type TOrdType

Value	Explanation
otSByte	Signed byte
otSLong	Signed longint
otSWord	Signed word
otUByte	Unsigned byte
otULong	Unsigned longint (Cardinal)
otUWord	Unsigned word

```
TTypeData = packed record
end
```

If the typeinfo kind is tkClass, then the property information follows the UnitName string, as an array of TPropInfo (1207) records.

```
TTypeInfo = record
  Kind : TTypeKind;
  Name : ShortString;
end
```

The TypeInfo function returns a pointer to a TTypeInfo record.

Note that the Name field is stored with as much bytes as needed to store the name, it is not padded to 255 characters. The type data immediately follows the TTypeInfo record as a TTypeData (1208) record.

```
TTypeKind = (tkUnknown,tkInteger,tkChar,tkEnumeration,tkFloat,tkSet,
             tkMethod,tkSString,tkLString,tkAString,tkWString,tkVariant,
             tkArray,tkRecord,tkInterface,tkClass,tkObject,tkWChar,
             tkBool,tkInt64,tkQWord,tkDynArray,tkInterfaceRaw)
```

Type of a property.

```
TTypeKinds= Set of (tkArray,tkAString,tkBool,tkChar,tkClass,tkDynArray,
                   tkEnumeration,tkFloat,tkInt64,tkInteger,tkInterface,
                   tkInterfaceRaw,tkLString,tkMethod,tkObject,tkQWord,
                   tkRecord,tkSet,tkSString,tkUnknown,tkVariant,
                   tkWChar,tkWString)
```

Set of TTypeKind (1208) enumeration.

```
Variant = Pointer
```

Dummy type. Do not use.

31.7 Procedures and functions

FindPropInfo

Synopsis: Return property information by property name.

Table 31.9: Enumeration values for type TTypeKind

Value	Explanation
tkArray	Array property.
tkAString	Ansistring property.
tkBool	Boolean property.
tkChar	Char property.
tkClass	Class property.
tkDynArray	Dynamical array property.
tkEnumeration	Enumeration type property.
tkFloat	Float property.
tkInt64	Int64 property.
tkInteger	Integer property.
tkInterface	Interface property.
tkInterfaceRaw	Raw interface property.
tkLString	Longstring property.
tkMethod	Method property.
tkObject	Object property.
tkQWord	QWord property.
tkRecord	Record property.
tkSet	Set property.
tkSString	Shortstring property.
tkUnknown	Unknown property type.
tkVariant	Variant property.
tkWChar	Widechar property.
tkWString	Widestring property.

Declaration: `function FindPropInfo(Instance: TObject;const PropName: String)
: PPropInfo
function FindPropInfo(AClass: TClass;const PropName: String) : PPropInfo`

Visibility: default

Description: FindPropInfo examines the published property information of a class and returns a pointer to the property information for property PropName. The class to be examined can be specified in one of two ways:

AClass a class pointer.

Instance an instance of the class to be investigated.

If the property does not exist, a EPropertyError exception will be raised. The GetPropInfo (1218) function has the same function as the FindPropInfo function, but returns Nil if the property does not exist.

Errors: Specifying an invalid property name in PropName will result in an EPropertyError exception.

See also: GetPropInfo (1218), GetPropList (1220), GetPropInfos (1219)

Listing: ./typinfex/ex14.pp

Program example13;

{ This program demonstrates the FindPropInfo function }

```

{$mode objfpc}

uses
  rttiobj , typinfo , sysutils ;

Var
  O : TMyTestObject ;
  PT : PTypeData ;
  PI : PPropInfo ;
  I , J : Longint ;
  PP : PPropList ;
  prl : PPropInfo ;

begin
  O:=TMyTestObject.Create ;
  PI:=FindPropInfo (O, 'BooleanField') ;
  WriteLn ( 'FindPropInfo (Instance , BooleanField) : ', PI^.Name ) ;
  PI:=FindPropInfo (O.ClassType, 'ByteField') ;
  WriteLn ( 'FindPropInfo (Class , ByteField) : ', PI^.Name ) ;
  Write ( 'FindPropInfo (Class , NonExistingProp) : ' ) ;
  Try
    PI:=FindPropInfo (O, 'NonExistingProp') ;
  except
    On E: Exception do
      WriteLn ( 'Caught exception " ', E.ClassName, '" with message : ', E.Message ) ;
    end ;
  O.Free ;
end .

```

GetEnumName

Synopsis: Return name of enumeration constant.

Declaration: `function GetEnumName (TypeInfo: PTypeInfo; Value: Integer) : String`

Visibility: default

Description: `GetEnumName` scans the type information for the enumeration type described by `TypeInfo` and returns the name of the enumeration constant for the element with ordinal value equal to `Value`.

If `Value` is out of range, the first element of the enumeration type is returned. The result is lower-cased, but this may change in the future.

This can be used in combination with `GetOrdProp` to stream a property of an enumerated type.

Errors: No check is done to determine whether `TypeInfo` really points to the type information for an enumerated type.

See also: `GetOrdProp` ([1217](#)), `GetEnumValue` ([1212](#))

Listing: `./typinfex/ex9.pp`

```

program example9 ;

{ This program demonstrates the GetEnumName, GetEnumValue functions }

{$mode objfpc}

```

```

WriteIn ( 'Get ( propinfo )           : ', GetEnumProp (O, PI) );
SetEnumProp (O, 'MyEnumField', 'meFirst' );
WriteIn ( 'Set ( name, meFirst )       : ', GetEnumName ( TI, Ord (O. MyEnumField) ));
SetEnumProp (O, PI, 'meSecond' );
WriteIn ( 'Set ( propinfo, meSecond ) : ', GetEnumName ( TI, Ord (O. MyEnumField) ));
O. Free;
end.

```

GetEnumValue

Synopsis: Get ordinal value for enumerated type by name

Declaration: `function GetEnumValue (TypeInfo: PTypeInfo; const Name: String) : Integer`

Visibility: default

Description: `GetEnumValue` scans the type information for the enumeration type described by `TypeInfo` and returns the ordinal value for the element in the enumerated type that has identifier `Name`. The identifier is searched in a case-insensitive manner.

This can be used to set the value of enumerated properties from a stream.

For an example, see `GetEnumName` ([1210](#)).

Errors: If `Name` is not found in the list of enumerated values, then -1 is returned. No check is done whether `TypeInfo` points to the type information for an enumerated type.

See also: `GetEnumName` ([1210](#)), `SetOrdProp` ([1229](#))

GetFloatProp

Synopsis: Return value of floating point property

Declaration: `function GetFloatProp (Instance: TObject; PropInfo: PPropInfo) : Extended`
`function GetFloatProp (Instance: TObject; const PropName: String)`
`: Extended`

Visibility: default

Description: `GetFloatProp` returns the value of the float property described by `PropInfo` or with name `Propname` for the object `Instance`. All float types are converted to extended.

Errors: No checking is done whether `Instance` is non-nil, or whether `PropInfo` describes a valid float property of `Instance`. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: `SetFloatProp` ([1228](#)), `GetOrdProp` ([1217](#)), `GetStrProp` ([1223](#)), `GetInt64Prop` ([1213](#)), `GetMethodProp` ([1214](#)), `GetSetProp` ([1221](#)), `GetObjectProp` ([1216](#)), `GetEnumProp` ([1211](#))

Listing: `./typinfex/ex4.pp`

```

program example4;

{ This program demonstrates the GetFloatProp function }

{$mode objfpc}

uses rttiobj, typinfo;

```

Var

```
O : TMyTestObject;
PI : PPropInfo;
```

begin

```
O:=TMyTestObject.Create;
Writeln('Real property : ');
PI:=GetPropInfo(O, 'RealField');
Writeln('Value           : ',O.RealField);
Writeln('Get (name)       : ',GetFloatProp(O, 'RealField'));
Writeln('Get (propinfo)      : ',GetFloatProp(O, PI));
SetFloatProp(O, 'RealField',system.Pi);
Writeln('Set (name,pi)       : ',O.RealField);
SetFloatProp(O, PI,exp(1));
Writeln('Set (propinfo,e)    : ',O.RealField);
Writeln('Extended property : ');
PI:=GetPropInfo(O, 'ExtendedField');
Writeln('Value           : ',O.ExtendedField);
Writeln('Get (name)       : ',GetFloatProp(O, 'ExtendedField'));
Writeln('Get (propinfo)    : ',GetFloatProp(O, PI));
SetFloatProp(O, 'ExtendedField',system.Pi);
Writeln('Set (name,pi)     : ',O.ExtendedField);
SetFloatProp(O, PI,exp(1));
Writeln('Set (propinfo,e)  : ',O.ExtendedField);
O.Free;
end.
```

GetInt64Prop

Synopsis: return value of an Int64 property

Declaration: `function GetInt64Prop(Instance: TObject;PropInfo: PPropInfo) : Int64`
`function GetInt64Prop(Instance: TObject;const PropName: String) : Int64`

Visibility: default

Description: Publishing of Int64 properties is not yet supported by Free Pascal. This function is provided for Delphi compatibility only at the moment.

GetInt64Prop returns the value of the property of type Int64 that is described by PropInfo or with name Propname for the object Instance.

Errors: No checking is done whether Instance is non-nil, or whether PropInfo describes a valid Int64 property of Instance. Specifying an invalid property name in PropName will result in an EPropertyError exception

See also: SetInt64Prop ([1228](#)), GetOrdProp ([1217](#)), GetStrProp ([1223](#)), GetFloatProp ([1212](#)), GetMethodProp ([1214](#)), GetSetProp ([1221](#)), GetObjectProp ([1216](#)), GetEnumProp ([1211](#))

Listing: ./typinfex/ex15.pp

```
program example15;

{ This program demonstrates the GetInt64Prop function }

{$mode objfpc}
```

```

uses rttiobj , typinfo ;

Var
  O : TMyTestObject;
  PI : PPropInfo;

begin
  O:=TMyTestObject.Create;
  Writeln('Int64 property : ');
  PI:=GetPropInfo(O, 'Int64Field');
  Writeln('Value           : ',O.Int64Field);
  Writeln('Get (name)       : ',GetInt64Prop(O, 'Int64Field'));
  Writeln('Get (propinfo)    : ',GetInt64Prop(O, PI));
  SetInt64Prop(O, 'Int64Field',12345);
  Writeln('Set (name,12345)  : ',O.Int64Field);
  SetInt64Prop(O, PI,54321);
  Writeln('Set (propinfo ,54321) : ',O.Int64Field);
  O.Free;
end.

```

GetMethodProp

Synopsis: Return value of a method property

Declaration: `function GetMethodProp(Instance: TObject;PropInfo: PPropInfo) : TMethod`
`function GetMethodProp(Instance: TObject;const PropName: String)`
`: TMethod`

Visibility: default

Description: `GetMethodProp` returns the method the property described by `PropInfo` or with name `Propname` for object `Instance`. The return type `TMethod` is defined in the `SysUtils` unit as:

```

TMethod = packed record
  Code, Data: Pointer;
end;

```

Data points to the instance of the class with the method Code.

Errors: No checking is done whether `Instance` is non-nil, or whether `PropInfo` describes a valid method property of `Instance`. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: `SetMethodProp` ([1228](#)), `GetOrdProp` ([1217](#)), `GetStrProp` ([1223](#)), `GetFloatProp` ([1212](#)), `GetInt64Prop` ([1213](#)), `GetSetProp` ([1221](#)), `GetObjectProp` ([1216](#)), `GetEnumProp` ([1211](#))

Listing: `./typinfex/ex6.pp`

```

program example6;

{ This program demonstrates the GetMethodProp function }

{$mode objfpc}

uses rttiobj , typinfo , sysutils;

```

Type

```

TNotifyObject = Class(TObject)
  Procedure Notification1(Sender : TObject);
  Procedure Notification2(Sender : TObject);
end;

```

```

Procedure TNotifyObject.Notification1(Sender : TObject);

```

begin

```

  Write('Received notification 1 of object with class: ');
  WriteLn(Sender.ClassName);

```

```

end;

```

```

Procedure TNotifyObject.Notification2(Sender : TObject);

```

begin

```

  Write('Received notification 2 of object with class: ');
  WriteLn(Sender.ClassName);

```

```

end;

```

Var

```

O : TMyTestObject;
PI : PPropInfo;
NO : TNotifyObject;
M : TMethod;

```

```

Procedure PrintMethod (Const M : TMethod);

```

begin

```

  If (M.Data=Pointer(NO)) Then
    If (M.Code=Pointer(@TNotifyObject.Notification1)) then
      WriteLn('Notification1')
    else If (M.Code=Pointer(@TNotifyObject.Notification2)) then
      WriteLn('Notification2')
    else
      begin
        Write('Unknown method adress (data: ');
        Write(hexStr(Longint(M.data),8));
        WriteLn(' ,code: ',hexstr(Longint(M.Code),8),')');
      end;

```

```

end;

```

begin

```

O:=TMyTestObject.Create;
NO:=TNotifyObject.Create;
O.NotifyEvent:=@NO.Notification1;
PI:=GetPropInfo(O,'NotifyEvent');
WriteLn('Method property : ');
Write('Notifying : ');
O.Notify;
Write('Get (name) : ');
M:=GetMethodProp(O,'NotifyEvent');
PrintMethod(M);
Write('Notifying : ');
O.Notify;
Write('Get (propinfo) : ');
M:=GetMethodProp(O,PI);

```

```

PrintMethod(M);
M.Data:=No;
M.Code:=Pointer(@NO.Notification2);
SetMethodProp(O,'NotifyEvent',M);
Write('Set (name, Notification2)      : ');
M:=GetMethodProp(O,PI);
PrintMethod(M);
Write('Notifying                        : ');
O.Notify;
Write('Set (propinfo, Notification1) : ');
M.Data:=No;
M.Code:=Pointer(@NO.Notification1);
SetMethodProp(O,PI,M);
M:=GetMethodProp(O,PI);
PrintMethod(M);
Write('Notifying                        : ');
O.Notify;
O.Free;
end.

```

GetObjectProp

Synopsis: Return value of an object-type property.

Declaration: `function GetObjectProp(Instance: TObject;const PropName: String) : TObject`

```

function GetObjectProp(Instance: TObject;const PropName: String;
MinClass: TClass) : TObject
function GetObjectProp(Instance: TObject;PropInfo: PPropInfo) : TObject
function GetObjectProp(Instance: TObject;PropInfo: PPropInfo;
MinClass: TClass) : TObject

```

Visibility: default

Description: `GetObjectProp` returns the object which the property described by `PropInfo` with name `Propname` points to for object `Instance`.

If `MinClass` is specified, then if the object is not descendent of class `MinClass`, then `Nil` is returned.

Errors: No checking is done whether `Instance` is non-nil, or whether `PropInfo` describes a valid method property of `Instance`. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: `SetMethodProp` (1228), `GetOrdProp` (1217), `GetStrProp` (1223), `GetFloatProp` (1212), `GetInt64Prop` (1213), `GetSetProp` (1221), `GetObjectProp` (1216), `GetEnumProp` (1211)

Listing: `./typinfex/ex5.pp`

program example5;

{ This program demonstrates the GetObjectProp function }

{ \$mode objfpc }

uses rttiobj, typinfo;

```

Var
  O : TMyTestObject;
  PI : PPropInfo;
  NO1,NO2 : TNamedObject;

begin
  O:=TMyTestObject.Create;
  NO1:=TNamedObject.Create;
  NO1.ObjectName:= 'First named object';
  NO2:=TNamedObject.Create;
  NO2.ObjectName:= 'Second named object';
  O.ObjField:=NO1;
  WriteIn( 'Object property : ');
  PI:=GetPropInfo(O, 'ObjField');
  Write( 'Property class      : ');
  WriteIn(GetObjectPropClass(O, 'ObjField').ClassName);
  Write( 'Value                : ');
  WriteIn((O.ObjField as TNamedObject).ObjectName);
  Write( 'Get (name)           : ');
  WriteIn((GetObjectProp(O, 'ObjField') As TNamedObject).ObjectName);
  Write( 'Get (propinfo)       : ');
  WriteIn((GetObjectProp(O,PI,TObject) as TNamedObject).ObjectName);
  SetObjectProp(O, 'ObjField',NO2);
  Write( 'Set (name,NO2)       : ');
  WriteIn((O.ObjField as TNamedObject).ObjectName);
  SetObjectProp(O,PI,NO1);
  Write( 'Set (propinfo,NO1) : ');
  WriteIn((O.ObjField as TNamedObject).ObjectName);
  O.Free;
end.

```

GetObjectPropClass

Synopsis: Return class of property.

Declaration: `function GetObjectPropClass(Instance: TObject;const PropName: String)
: TClass`

Visibility: default

Description: `GetObjectPropClass` returns the declared class of the property with name `PropName`. This may not be the actual class of the property value.

For an example, see `GetObjectProp` ([1216](#)).

Errors: No checking is done whether `Instance` is non-nil. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: `SetMethodProp` ([1228](#)), `GetOrdProp` ([1217](#)), `GetStrProp` ([1223](#)), `GetFloatProp` ([1212](#)), `GetInt64Prop` ([1213](#))

GetOrdProp

Synopsis: Get the value of an ordinal property

Declaration: `function GetOrdProp(Instance: TObject;PropInfo: PPropInfo) : LongInt`
`function GetOrdProp(Instance: TObject;const PropName: String) : LongInt`

Visibility: default

Description: GetOrdProp returns the value of the ordinal property described by PropInfo or with name PropName for the object Instance. The value is returned as a longint, which should be typecasted to the needed type.

Ordinal properties that can be retrieved include:

Integers and subranges of integersThe value of the integer will be returned.

Enumerated types and subranges of enumerated typesThe ordinal value of the enumerated type will be returned.

Sets If the base type of the set has less than 31 possible values. If a bit is set in the return value, then the corresponding element of the base ordinal class of the set type must be included in the set.

Errors: No checking is done whether `Instance` is non-nil, or whether `PropInfo` describes a valid ordinal property of `Instance`. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: [SetOrdProp \(1229\)](#), [GetStrProp \(1223\)](#), [GetFloatProp \(1212\)](#), [GetInt64Prop \(1213\)](#), [GetMethodProp \(1214\)](#), [GetSetProp \(1221\)](#), [GetObjectProp \(1216\)](#), [GetEnumProp \(1211\)](#)

Listing: ./typinfex/ex1.pp

```

program example1;

{ This program demonstrates the GetOrdProp function }

{$mode objfpc}

uses rttiobj, typinfo;

Var
    O : TMyTestObject;
    PI : PPropInfo;

begin
    O:=TMyTestObject.Create;
    WriteLn( 'Boolean property      : ');
    WriteLn( 'Value                  : ',O.BooleanField);
    WriteLn( 'Ord(Value)              : ',Ord(O.BooleanField));
    WriteLn( 'Get (name)              : ',GetOrdProp(O, 'BooleanField' ));
    PI:=GetPropInfo(O, 'BooleanField' );
    WriteLn( 'Get (propinfo)           : ',GetOrdProp(O, PI ));
    SetOrdProp(O, 'BooleanField',Ord(False));
    WriteLn( 'Set (name,false)         : ',O.BooleanField);
    SetOrdProp(O, PI, Ord(True));
    WriteLn( 'Set (propinfo, true)        : ',O.BooleanField);
    O.Free;
end.

```

GetPropInfo

Synopsis: Return property type information, by property name.

```
Declaration: function GetPropInfo(TypeInfo: PTypeInfo;const PropName: String)
                        : PPropInfo
```

```

function GetPropInfo(TypeInfo: PTypeInfo;const PropName: String;
                    AKinds: TTypeKinds) : PPropInfo
function GetPropInfo(Instance: TObject;const PropName: String;
                    AKinds: TTypeKinds) : PPropInfo
function GetPropInfo(Instance: TObject;const PropName: String)
                    : PPropInfo
function GetPropInfo(AClass: TClass;const PropName: String;
                    AKinds: TTypeKinds) : PPropInfo
function GetPropInfo(AClass: TClass;const PropName: String) : PPropInfo

```

Visibility: default

Description: GetPropInfo returns a pointer to the TPropInfo record for a the PropName property of a class. The class to examine can be specified in one of three ways:

InstanceAn instance of the class.

AClassA class pointer to the class.

TypeInfoA pointer to the type information of the class.

In each of these three ways, if AKinds is specified, if the property has TypeKind which is not included in AKinds, Nil will be returned.

For an example, see most of the other functions.

Errors: If the property PropName does not exist, Nil is returned.

See also: GetPropInfos ([1219](#)), GetPropList ([1220](#))

GetPropInfos

Synopsis: Return a list of published properties.

Declaration: procedure GetPropInfos(TypeInfo: PTypeInfo;PropList: PPropList)

Visibility: default

Description: GetPropInfos stores pointers to the property information of all published properties of a class with class info TypeInfo in the list pointed to by PropList. The PropList pointer must point to a memory location that contains enough space to hold all properties of the class and its parent classes.

Errors: No checks are done to see whether PropList points to a memory area that is big enough to hold all pointers.

See also: GetPropInfo ([1218](#)), GetPropList ([1220](#))

Listing: ./typinfex/ex12.pp

Program example12;

{ This program demonstrates the GetPropInfos function }

uses

rttiobj, typinfo;

Var

O : TMyTestObject;

```

PT : PTypeData;
PI : PTypeInfo;
I,J : Longint;
PP : PPropList;
prl : PPropInfo;

begin
O:=TMyTestObject.Create;
PI:=O.ClassInfo;
PT:=GetTypeData(PI);
WriteLn('Property Count : ',PT^.PropCount);
GetMem (PP,PT^.PropCount*SizeOf(Pointer));
GetPropInfos(PI,PP);
For I:=0 to PT^.PropCount-1 do
begin
With PP^[I]^ do
begin
Write('Property ',I+1:3,' : ',name:30);
writeln(' Type: ',TypeNames[typinfo.PropType(O,Name)]);
end;
end;
FreeMem(PP);
O.Free;
end.

```

GetPropList

Synopsis: Return a list of a certain type of published properties.

Declaration: `function GetPropList(TypeInfo: PTypeInfo;TypeKinds: TTypeKinds; PropList: PPropList) : Integer`

Visibility: default

Description: `GetPropList` stores pointers to property information of the class with class info `TypeInfo` for properties of kind `TypeKinds` in the list pointed to by `PropList`. `PropList` must contain enough space to hold all properties.

The function returns the number of pointers that matched the criteria and were stored in `PropList`.

Errors: No checks are done to see whether `PropList` points to a memory area that is big enough to hold all pointers.

See also: `GetPropInfos` ([1219](#)), `GetPropInfo` ([1218](#))

Listing: ./typinfex/ex13.pp

Program example13;

{ This program demonstrates the GetPropList function }

uses

rttiobj, typinfo;

Var

O : TMyTestObject;

PT : PTypeData;

```

PI : PTypeInfo;
I,J : Longint;
PP : PPropList;
prl : PPropInfo;

begin
  O:=TMyTestObject.Create;
  PI:=O.ClassInfo;
  PT:=GetTypeData(PI);
  WriteLn('Total property Count : ',PT^.PropCount);
  GetMem (PP,PT^.PropCount*SizeOf(Pointer));
  J:=GetPropList(PI,OrdinalTypes,PP);
  WriteLn('Ordinal property Count : ',J);
  For I:=0 to J-1 do
    begin
      With PP^[i]^ do
        begin
          Write('Property ',i+1:3,' : ',name:30);
          writeln(' Type: ',TypeNames[typinfo.PropType(O,Name)]);
        end;
      end;
    FreeMem(PP);
  O.Free;
end.

```

GetPropValue

Synopsis: Get property value as a string.

Declaration: `function GetPropValue(Instance: TObject;const PropName: String) : Variant`
`function GetPropValue(Instance: TObject;const PropName: String;
 PreferStrings: Boolean) : Variant`

Visibility: default

Description: Due to missing Variant support, GetPropValue is not yet implemented. The declaration is provided for compatibility with Delphi.

Errors:

GetSetProp

Synopsis: Return the value of a set property.

Declaration: `function GetSetProp(Instance: TObject;const PropName: String) : String`
`function GetSetProp(Instance: TObject;const PropName: String;
 Brackets: Boolean) : String`
`function GetSetProp(Instance: TObject;const PropInfo: PPropInfo;
 Brackets: Boolean) : String`

Visibility: default

Description: GetSetProp returns the contents of a set property as a string. The property to be returned can be specified by it's name in PropName or by its property information in PropInfo.

The returned set is a string representation of the elements in the set as returned by SetToString ([1231](#)). The Brackets option can be used to enclose the string representation in square brackets.

Errors: No checking is done whether Instance is non-nil, or whether PropInfo describes a valid ordinal property of Instance. Specifying an invalid property name in PropName will result in an EPropertyError exception.

See also: SetSetProp (1230), GetStrProp (1223), GetFloatProp (1212), GetInt64Prop (1213), GetMethodProp (1214)

Listing: ./typinfex/ex7.pp

```

program example7;

{ This program demonstrates the GetSetProp function }

{$mode objfpc}

uses rttiobj , typinfo;

Var
  O : TMyTestObject;
  PI : PPropInfo;

Function SetAsString (ASet : TMyEnums) : String;

Var
  i : TmyEnum;

begin
  result := '';
  For i := mefirst to methird do
    If i in ASet then
      begin
        If (Result <> '') then
          Result := Result + ', ';
          Result := Result + MyEnumNames[i];
        end;
      end;

Var
  S : TMyEnums;

begin
  O := TMyTestObject.Create;
  O.SetField := [mefirst, meSecond, meThird];
  Writeln ('Set property      : ');
  Writeln ('Value                               : ', SetAsString(O.SetField));
  Writeln ('Ord(Value)                             : ', Longint(O.SetField));
  Writeln ('Get (name)                               : ', GetSetProp(O, 'SetField'));
  PI := GetPropInfo(O, 'SetField');
  Writeln ('Get (propinfo)                           : ', GetSetProp(O, PI, false));
  S := [meFirst, meThird];
  SetOrdProp(O, 'SetField', Integer(S));
  Write ('Set (name,[mefirst, methird]) : ');
  Writeln (SetAsString(O.SetField));
  S := [meSecond];
  SetOrdProp(O, PI, Integer(S));
  Write ('Set (propinfo,[meSecond])      : ');
  Writeln (SetAsString(O.SetField));
  O.Free;
end.

```

GetStrProp

Synopsis: Return the value of a string property.

Declaration: `function GetStrProp(Instance: TObject; PropInfo: PPropInfo) : Ansistring`
`function GetStrProp(Instance: TObject; const PropName: String) : String`

Visibility: default

Description: `GetStrProp` returns the value of the string property described by `PropInfo` or with name `PropName` for object `Instance`.

Errors: No checking is done whether `Instance` is non-nil, or whether `PropInfo` describes a valid string property of `Instance`. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: `SetStrProp` (1231), `GetOrdProp` (1217), `GetFloatProp` (1212), `GetInt64Prop` (1213), `GetMethodProp` (1214)

Listing: `./typinfex/ex3.pp`

```

program example3;

{ This program demonstrates the GetStrProp function }

{$mode objfpc}

uses rttiobj, typinfo;

Var
  O : TMyTestObject;
  PI : PPropInfo;

begin
  O := TMyTestObject.Create;
  PI := GetPropInfo(O, 'AnsiStringField');
  Writeln('String property : ');
  Writeln('Value           : ', O.AnsiStringField);
  Writeln('Get (name)         : ', GetStrProp(O, 'AnsiStringField'));
  Writeln('Get (propinfo)        : ', GetStrProp(O, PI));
  SetStrProp(O, 'AnsiStringField', 'First');
  Writeln('Set (name, ''First'')   : ', O.AnsiStringField);
  SetStrProp(O, PI, 'Second');
  Writeln('Set (propinfo, ''Second'') : ', O.AnsiStringField);
  O.Free;
end.

```

GetTypeData

Synopsis: Return a pointer to type data, based on type information.

Declaration: `function GetTypeData(TypeInfo: PTypeInfo) : PTypeData`

Visibility: default

Description: `GetTypeData` returns a pointer to the `TTypeData` record that follows after the `TTypeInfo` record pointed to by `TypeInfo`. It essentially skips the `Kind` and `Name` fields in the `TTypeInfo` record.

Errors: None.

GetVariantProp

Synopsis: Return the value of a variant property.

Declaration: `function GetVariantProp(Instance: TObject; PropInfo: PPropInfo) : Variant`
`function GetVariantProp(Instance: TObject; const PropName: String)`
`: Variant`

Visibility: default

Description: Due to missing Variant support, the GetVariantProp function is not yet implemented. Provided for Delphi compatibility only.

Errors:

See also: SetVariantProp ([1232](#))

IsPublishedProp

Synopsis: Check whether a published property exists.

Declaration: `function IsPublishedProp(Instance: TObject; const PropName: String)`
`: Boolean`
`function IsPublishedProp(AClass: TClass; const PropName: String)`
`: Boolean`

Visibility: default

Description: IsPublishedProp returns true if a class has a published property with name PropName. The class can be specified in one of two ways:

AClass A class pointer to the class.

Instance An instance of the class.

Errors: No checks are done to ensure Instance or AClass are valid pointers. Specifying an invalid property name in PropName will result in an EPropertyError exception.

See also: IsStoredProp ([1225](#)), PropIsType ([1226](#))

Listing: ./typinfex/ex10.pp

```

program example10;

{ This program demonstrates the IsPublishedProp function }

{$mode objfpc}

uses rttiobj, typinfo;

Var
  O : TMyTestObject;
  PI : PPropInfo;

begin
  O:=TMyTestObject.Create;
  WriteLn( 'Property tests      : ');
  Write( 'IsPublishedProp(O, BooleanField)      : ');
  WriteLn(IsPublishedProp(O, 'BooleanField'));
  Write( 'IsPublishedProp(Class, BooleanField) : ');

```

```

WriteIn (IsPublishedProp (O.ClassType, 'BooleanField'));
Write ( 'IsPublishedProp (O, SomeField)      : ');
WriteIn (IsPublishedProp (O, 'SomeField'));
Write ( 'IsPublishedProp (Class, SomeField)   : ');
WriteIn (IsPublishedProp (O.ClassType, 'SomeField'));
O.Free;
end.

```

IsStoredProp

Synopsis: Check whether a property is stored.

Declaration: `function IsStoredProp (Instance: TObject; PropInfo: PPropInfo) : Boolean`
`function IsStoredProp (Instance: TObject; const PropName: String)`
`: Boolean`

Visibility: default

Description: `IsStoredProp` returns `True` if the `Stored` modifier evaluates to `True` for the property described by `PropInfo` or with name `PropName` for object `Instance`. It returns `False` otherwise. If the function returns `True`, this indicates that the property should be written when streaming the object `Instance`.

If there was no `stored` modifier in the declaration of the property, `True` will be returned.

Errors: No checking is done whether `Instance` is non-nil, or whether `PropInfo` describes a valid property of `Instance`. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: `IsPublishedProp` ([1224](#)), `PropIsType` ([1226](#))

Listing: `./typinfex/ex11.pp`

```

program example11;

{ This program demonstrates the IsStoredProp function }

{$mode objfpc}

uses rttiobj, typinfo;

Var
  O : TMyTestObject;
  PI : PPropInfo;

begin
  O:=TMyTestObject.Create;
  WriteIn ('Stored tests      : ');
  Write ('IsStoredProp (O, StoredIntegerConstFalse) : ');
  WriteIn (IsStoredProp (O, 'StoredIntegerConstFalse'));
  Write ('IsStoredProp (O, StoredIntegerConstTrue)   : ');
  WriteIn (IsStoredProp (O, 'StoredIntegerConstTrue'));
  Write ('IsStoredProp (O, StoredIntegerMethod)      : ');
  WriteIn (IsStoredProp (O, 'StoredIntegerMethod'));
  Write ('IsStoredProp (O, StoredIntegerVirtualMethod) : ');
  WriteIn (IsStoredProp (O, 'StoredIntegerVirtualMethod'));
  O.Free;
end.

```

PropIsType

Synopsis: Check the type of a published property.

Declaration: `function PropIsType(Instance: TObject;const PropName: String;
TypeKind: TTypeKind) : Boolean`
`function PropIsType(AClass: TClass;const PropName: String;
TypeKind: TTypeKind) : Boolean`

Visibility: default

Description: `PropIsType` returns `True` if the property with name `PropName` has type `TypeKind`. It returns `False` otherwise. The class to be examined can be specified in one of two ways:

AClassA class pointer.

InstanceAn instance of the class.

Errors: No checks are done to ensure `Instance` or `AClass` are valid pointers. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: `IsPublishedProp` ([1224](#)), `IsStoredProp` ([1225](#)), `PropType` ([1226](#))

Listing: `./typinfex/ex16.pp`

```
program example16;

{ This program demonstrates the PropIsType function }

{$mode objfpc}

uses rttiobj, typinfo;

Var
  O : TMyTestObject;

begin
  O:= TMyTestObject.Create;
  Writeln('Property tests      : ');
  Write('PropIsType(O, BooleanField, tkBool)      : ');
  Writeln(PropIsType(O, 'BooleanField', tkBool));
  Write('PropIsType(Class, BooleanField, tkBool) : ');
  Writeln(PropIsType(O.ClassType, 'BooleanField', tkBool));
  Write('PropIsType(O, ByteField, tkString)      : ');
  Writeln(PropIsType(O, 'ByteField', tkString));
  Write('PropIsType(Class, ByteField, tkString) : ');
  Writeln(PropIsType(O.ClassType, 'ByteField', tkString));
  O.Free;
end.
```

PropType

Synopsis: Return the type of a property

Declaration: `function PropType(Instance: TObject;const PropName: String) : TTypeKind`
`function PropType(AClass: TClass;const PropName: String) : TTypeKind`

Visibility: default

Description: `PropType` returns the type of the property `PropName` for a class. The class to be examined can be specified in one of 2 ways:

AClass A class pointer.

Instance An instance of the class.

Errors: No checks are done to ensure `Instance` or `AClass` are valid pointers. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: `IsPublishedProp` ([1224](#)), `IsStoredProp` ([1225](#)), `PropIsType` ([1226](#))

Listing: `./typinfex/ex17.pp`

```

program example17;

{ This program demonstrates the PropType function }

{$mode objfpc}

uses rttiobj, typinfo;

Var
  O : TMyTestObject;

begin
  O := TMyTestObject.Create;
  Writeln ('Property tests      : ');
  Write ('PropType(O, BooleanField)      : ');
  Writeln (TypeNames[PropType(O, 'BooleanField')]);
  Write ('PropType(Class, BooleanField) : ');
  Writeln (TypeNames[PropType(O.ClassType, 'BooleanField')]);
  Write ('PropType(O, ByteField)      : ');
  Writeln (TypeNames[PropType(O, 'ByteField')]);
  Write ('PropType(Class, ByteField)    : ');
  Writeln (TypeNames[PropType(O.ClassType, 'ByteField')]);
  O.Free;
end.

```

SetEnumProp

Synopsis: Set value of an enumerated-type property

Declaration:

```

procedure SetEnumProp(Instance: TObject; const PropName: String;
                     const Value: String)
procedure SetEnumProp(Instance: TObject; const PropInfo: PPropInfo;
                     const Value: String)

```

Visibility: default

Description: `SetEnumProp` sets the property described by `PropInfo` or with name `PropName` to `Value`. `Value` must be a string with the name of the enumerate value, i.e. it can be used as an argument to `GetEnumValue` ([1212](#)).

For an example, see `GetEnumProp` ([1211](#)).

Errors: No checks are done to ensure `Instance` or `PropInfo` are valid pointers. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: [GetEnumProp \(1211\)](#), [SetStrProp \(1231\)](#), [SetFloatProp \(1228\)](#), [SetInt64Prop \(1228\)](#), [SetMethodProp \(1228\)](#)

SetFloatProp

Synopsis: Set value of a float property.

Declaration:

```
procedure SetFloatProp(Instance: TObject; const PropName: String;
                      Value: Extended)
procedure SetFloatProp(Instance: TObject; PropInfo: PPropInfo;
                      Value: Extended)
```

Visibility: default

Description: `SetFloatProp` assigns `Value` to the property described by `PropInfo` or with name `Propname` for the object `Instance`.

For an example, see [GetFloatProp \(1212\)](#).

Errors: No checking is done whether `Instance` is non-nil, or whether `PropInfo` describes a valid float property of `Instance`. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: [GetFloatProp \(1212\)](#), [SetOrdProp \(1229\)](#), [SetStrProp \(1231\)](#), [SetInt64Prop \(1228\)](#), [SetMethodProp \(1228\)](#)

SetInt64Prop

Synopsis: Set value of a Int64 property

Declaration:

```
procedure SetInt64Prop(Instance: TObject; PropInfo: PPropInfo;
                      const Value: Int64)
procedure SetInt64Prop(Instance: TObject; const PropName: String;
                      const Value: Int64)
```

Visibility: default

Description: `SetInt64Prop` assigns `Value` to the property of type `Int64` that is described by `PropInfo` or with name `Propname` for the object `Instance`.

For an example, see [GetInt64Prop \(1213\)](#).

Errors: No checking is done whether `Instance` is non-nil, or whether `PropInfo` describes a valid `Int64` property of `Instance`. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: [GetInt64Prop \(1213\)](#), [GetMethodProp \(1214\)](#), [SetOrdProp \(1229\)](#), [SetStrProp \(1231\)](#), [SetFloatProp \(1228\)](#)

SetMethodProp

Synopsis: Set the value of a method property

Declaration:

```
procedure SetMethodProp(Instance: TObject; PropInfo: PPropInfo;
                      const Value: TMethod)
procedure SetMethodProp(Instance: TObject; const PropName: String;
                      const Value: TMethod)
```

Visibility: default

Description: SetMethodProp assigns Value to the method the property described by PropInfo or with name Propname for object Instance.

The type TMethod of the Value parameter is defined in the SysUtils unit as:

```
TMethod = packed record
  Code, Data: Pointer;
end;
```

Data should point to the instance of the class with the method Code.

For an example, see GetMethodProp (1214).

Errors: No checking is done whether Instance is non-nil, or whether PropInfo describes a valid method property of Instance. Specifying an invalid property name in PropName will result in an EPropertyError exception.

See also: GetMethodProp (1214), SetOrdProp (1229), SetStrProp (1231), SetFloatProp (1228), SetInt64Prop (1228)

SetObjectProp

Synopsis: Set the value of an object-type property.

Declaration:

```
procedure SetObjectProp(Instance: TObject;const PropName: String;
  Value: TObject)
procedure SetObjectProp(Instance: TObject;PropInfo: PPropInfo;
  Value: TObject)
```

Visibility: default

Description: SetObjectProp assigns Value to the the object property described by PropInfo or with name Propname for the object Instance.

For an example, see GetObjectProp (1216).

Errors: No checking is done whether Instance is non-nil, or whether PropInfo describes a valid method property of Instance. Specifying an invalid property name in PropName will result in an EPropertyError exception.

See also: GetObjectProp (1216), SetOrdProp (1229), SetStrProp (1231), SetFloatProp (1228), SetInt64Prop (1228), SetMethodProp (1228)

SetOrdProp

Synopsis: Set value of an ordinal property

Declaration:

```
procedure SetOrdProp(Instance: TObject;PropInfo: PPropInfo;
  Value: LongInt)
procedure SetOrdProp(Instance: TObject;const PropName: String;
  Value: LongInt)
```

Visibility: default

Description: SetOrdProp assigns Value to the the ordinal property described by PropInfo or with name Propname for the object Instance.

Ordinal properties that can be set include:

Integers and subranges of integersThe actual value of the integer must be passed.

Enumerated types and subranges of enumerated typesThe ordinal value of the enumerated type must be passed.

Subrange typesof integers or enumerated types. Here the ordinal value must be passed.

SetsIf the base type of the set has less than 31 possible values. For each possible value; the corresponding bit of `Value` must be set.

For an example, see `GetOrdProp` (1217).

Errors: No checking is done whether `Instance` is non-nil, or whether `PropInfo` describes a valid ordinal property of `Instance`. No range checking is performed. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: `GetOrdProp` (1217), `SetStrProp` (1231), `SetFloatProp` (1228), `SetInt64Prop` (1228), `SetMethodProp` (1228)

SetPropValue

Synopsis: Set property value as variant

Declaration: `procedure SetPropValue(Instance: TObject;const PropName: String;
const Value: Variant)`

Visibility: default

Description: Due to missing Variant support, this function is not yet implemented; it is provided for Delphi compatibility only.

Errors:

SetSetProp

Synopsis: Set value of set-typed property.

Declaration: `procedure SetSetProp(Instance: TObject;const PropName: String;
const Value: String)
procedure SetSetProp(Instance: TObject;const PropInfo: PPropInfo;
const Value: String)`

Visibility: default

Description: `SetSetProp` sets the property specified by `PropInfo` or `PropName` for object `Instance` to `Value`. `Value` is a string which contains a comma-separated list of values, each value being a string-representation of the enumerated value that should be included in the set. The value should be accepted by the `StringToSet` (1232) function.

The value can be formed using the `SetToString` (1231) function.

For an example, see `GetSetProp` (1221).

Errors: No checking is done whether `Instance` is non-nil, or whether `PropInfo` describes a valid ordinal property of `Instance`. No range checking is performed. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: `GetSetProp` (1221), `SetOrdProp` (1229), `SetStrProp` (1231), `SetFloatProp` (1228), `SetInt64Prop` (1228), `SetMethodProp` (1228), `SetToString` (1231), `StringToSet` (1232)

SetStrProp

Synopsis: Set value of a string property

```

Declaration: procedure SetStrProp(Instance: TObject; const PropName: String;
                                const Value: AnsiString)
              procedure SetStrProp(Instance: TObject; PropInfo: PPropInfo;
                                const Value: Ansistring)

```

Visibility: default

Description: SetStrProp assigns Value to the string property described by PropInfo or with name Propname for object Instance.

For an example, see GetStrProp ([1223](#))

Errors: No checking is done whether Instance is non-nil, or whether PropInfo describes a valid string property of Instance. Specifying an invalid property name in PropName will result in an EPropertyError exception.

See also: GetStrProp ([1223](#)), SetOrdProp ([1229](#)), SetFloatProp ([1228](#)), SetInt64Prop ([1228](#)), SetMethodProp ([1228](#))

SetToString

Synopsis: Convert set to a string description

```

Declaration: function SetToString(PropInfo: PPropInfo; Value: Integer;
                                Brackets: Boolean) : String
              function SetToString(PropInfo: PPropInfo; Value: Integer) : String

```

Visibility: default

Description: SetToString takes an integer representation of a set (as received e.g. by GetOrdProp) and turns it into a string representing the elements in the set, based on the type information found in the PropInfo property information. By default, the string representation is not surrounded by square brackets. Setting the Brackets parameter to True will surround the string representation with brackets.

The function returns the string representation of the set.

Errors: No checking is done to see whether PropInfo points to valid property information.

See also: GetEnumName ([1210](#)), GetEnumValue ([1212](#)), StringToSet ([1232](#))

Listing: ./typinfex/ex18.pp

```

program example18;

{ This program demonstrates the SetToString function }

{$mode objfpc}

uses rttiobj, typinfo;

Var
  O : TMyTestObject;
  PI : PPropInfo;
  I : longint;

```

```

begin
  O:=TMyTestObject.Create;
  PI:=GetPropInfo(O, 'SetField');
  O.SetField :=[ mefirst ,meSecond, meThird];
  I:=GetOrdProp(O, PI);
  WriteLn('Set property to string : ');
  WriteLn('Value  : ',SetToString(PI,I,False));
  O.SetField :=[ mefirst ,meSecond];
  I:=GetOrdProp(O, PI);
  WriteLn('Value  : ',SetToString(PI,I,True));
  I:=StringToSet(PI, 'mefirst');
  SetOrdProp(O, PI, I);
  I:=GetOrdProp(O, PI);
  WriteLn('Value  : ',SetToString(PI,I,False));
  I:=StringToSet(PI, '[mesecond, methird]');
  SetOrdProp(O, PI, I);
  I:=GetOrdProp(O, PI);
  WriteLn('Value  : ',SetToString(PI,I,True));
  O.Free;
end.

```

SetVariantProp

Synopsis: Set value of a variant property

Declaration: `procedure SetVariantProp(Instance: TObject;const PropName: String;
const Value: Variant)
procedure SetVariantProp(Instance: TObject;PropInfo: PPropInfo;
const Value: Variant)`

Visibility: default

Description: Due to missing Variant support, this function is not yet implemented. Provided for Delphi compatibility only.

Errors:

StringToSet

Synopsis: Convert string description to a set.

Declaration: `function StringToSet(PropInfo: PPropInfo;const Value: String) : Integer`

Visibility: default

Description: `StringToSet` converts the string representation of a set in `Value` to a integer representation of the set, using the property information found in `PropInfo`. This property information should point to the property information of a set property. The function returns the integer representation of the set. (i.e, the set value, typecast to an integer)

The string representation can be surrounded with square brackets, and must consist of the names of the elements of the base type of the set. The base type of the set should be an enumerated type. The elements should be separated by commas, and may be surrounded by spaces. each of the names will be fed to the `GetEnumValue` ([1212](#)) function.

For an example, see `SetToString` ([1231](#)).

Errors: No checking is done to see whether `PropInfo` points to valid property information. If a wrong name is given for an enumerated value, then an `EPropertyError` will be raised.

See also: `GetEnumName` ([1210](#)), `GetEnumValue` ([1212](#)), `SetToString` ([1231](#))

31.8 EPropertyError

Description

Exception raised in case of an error in one of the functions.

Chapter 32

Reference for unit 'Unix'

32.1 Used units

Table 32.1: Used units by unit 'Unix'

Name	Page
BaseUnix	71
unixtype	1272

32.2 Constants, types and variables

Constants

`ARG_MAX = UnixType.ARG_MAX`

Maximum number of arguments to a program.

`fs_ext = $137d`

File system type (StatFS ([1269](#))): (ext) Extended

`fs_ext2 = $ef53`

File system type (StatFS ([1269](#))): (ext2) Second extended

`fs_iso = $9660`

File system type (StatFS ([1269](#))): ISO 9660

`fs_minix = $137f`

File system type (StatFS ([1269](#))): Minix

`fs_minix_30 = $138f`

File system type (StatFS (1269)): Minix 3.0

`fs_minix_V2` = \$2468

File system type (StatFS (1269)): Minix V2

`fs_msdos` = \$4d44

File system type (StatFS (1269)): MSDOS (FAT)

`fs_nfs` = \$6969

File system type (StatFS (1269)): NFS

`fs_old_ext2` = \$ef51

File system type (StatFS (1269)): (ext2) Old second extended

`fs_proc` = \$9fa0

File system type (StatFS (1269)): PROC fs

`fs_xia` = \$012FD16D

File system type (StatFS (1269)): XIA

`IOctl_TCGETS` = \$5401

IOCTL call number: get Terminal Control settings

`LOCK_EX` = 2

FpFLock (1260) Exclusive lock

`LOCK_NB` = 4

FpFLock (1260) Non-blocking operation

`LOCK_SH` = 1

FpFLock (1260) Shared lock

`LOCK_UN` = 8

FpFLock (1260) unlock

`MAP_ANONYMOUS` = \$20

`MAP_DENYWRITE` = \$800

`MAP_EXECUTABLE` = \$1000

MAP_FIXED = \$10

MAP_GROWSDOWN = \$100

MAP_LOCKED = \$2000

MAP_NORESERVE = \$4000

MAP_PRIVATE = 2

MAP_SHARED = \$1

MAP_TYPE = \$f

NAME_MAX = UnixType.NAME_MAX

Maximum filename length.

Open_Accmode = 3

Bitmask to determine access mode in open flags.

Open_Append = 2 shl 9

File open mode: Append to file

Open_Creat = 1 shl 6

File open mode: Create if file does not yet exist.

Open_Direct = 4 shl 12

File open mode: Minimize caching effects

Open_Directory = 2 shl 15

File open mode: File must be directory.

Open_Excl = 2 shl 6

File open mode: Open exclusively

Open_LargeFile = 1 shl 15

File open mode: Open for 64-bit I/O

Open_NDelay = Open_NonBlock

File open mode: Alias for `Open_NonBlock` ([1237](#))

`Open_NoCtty` = 4 shl 6

File open mode: No TTY control.

`Open_NoFollow` = 4 shl 15

File open mode: Fail if file is symbolic link.

`Open_NonBlock` = 4 shl 9

File open mode: Open in non-blocking mode

`Open_RdOnly` = 0

File open mode: Read only

`Open_RdWr` = 2

File open mode: Read/Write

`Open_Sync` = 1 shl 12

File open mode: Write to disc at once

`Open_Trunc` = 1 shl 9

File open mode: Truncate file to length 0

`Open_WrOnly` = 1

File open mode: Write only

`PATH_MAX` = `UnixType.PATH_MAX`

Maximum pathname length.

`PROT_EXEC` = \$4

`PROT_NONE` = \$0

`PROT_READ` = \$1

`PROT_WRITE` = \$2

`P_IN` = 1

Input file descriptor of pipe pair.

`P_OUT` = 2

Output file descriptor of pipe pair.

SA_INTERRUPT = \$20000000

Sigaction options: ?

SA_NOCLDSTOP = 1

Sigaction options: Do not receive notification when child processes stop

SA_NOCLDWAIT = 2

Sigaction options: ?

SA_NOMASK = \$40000000

Sigaction options: Do not prevent the signal from being received when it is handled.

SA_ONESHOT = \$80000000

Sigaction options: Restore the signal action to the default state.

SA_RESTART = \$10000000

Sigaction options: Provide behaviour compatible with BSD signal semantics

SA_SHIRQ = \$04000000

Sigaction options: ?

SA_SIGINFO = 4

Sigaction options: The signal handler takes 3 arguments, not one.

SA_STACK = \$08000000

Sigaction options: Call the signal handler on an alternate signal stack.

Seek_Cur = 1

Seek option: Set position relative to current position.

Seek_End = 2

Seek option: Set position relative to end of file.

Seek_set = 0

Seek option: Set absolute position.

SIGABRT = 6

Signal: ABRT (Abort)

SIGALRM = 14

Signal: ALRM (Alarm clock)

SIGBUS = 7

Signal: BUS (bus error)

SIGCHLD = 17

Signal: CHLD (child status changed)

SIGCONT = 18

Signal: CONT (Continue)

SIGFPE = 8

Signal: FPE (Floating point error)

SIGHUP = 1

Signal: HUP (Hangup)

SIGILL = 4

Signal: ILL (Illegal instruction)

SIGINT = 2

Signal: INT (Interrupt)

SIGIO = 29

Signal: IO (I/O operation possible)

SIGIOT = 6

Signal: IOT (IOT trap)

SIGKILL = 9

Signal: KILL (unblockable)

SIGPIPE = 13

Signal: PIPE (Broken pipe)

SIGPOLL = SIGIO

Signal: POLL (Pollable event)

SIGPROF = 27

Signal: PROF (Profiling alarm)

SIGPWR = 30

Signal: PWR (power failure restart)

SIGQUIT = 3

Signal: QUIT

SIGSEGV = 11

Signal: SEGV (Segmentation violation)

SIGSTKFLT = 16

Signal: STKFLT (Stack Fault)

SIGSTOP = 19

Signal: STOP (Stop, unblockable)

SIGTerm = 15

Signal: TERM (Terminate)

SIGTRAP = 5

Signal: TRAP (Trace trap)

SIGTSTP = 20

Signal: TSTP (keyboard stop)

SIGTTIN = 21

Signal: TTIN (Terminal input, background)

SIGTTOU = 22

Signal: TTOU (Terminal output, background)

SIGUNUSED = 31

Signal: Unused

SIGURG = 23

Signal: URG (Socket urgent condition)

SIGUSR1 = 10

Signal: USR1 (User-defined signal 1)

SIGUSR2 = 12

Signal: USR2 (User-defined signal 2)

SIGVTALRM = 26

Signal: VTALRM (Virtual alarm clock)

SIGWINCH = 28

Signal: WINCH (Window/Terminal size change)

SIGXCPU = 24

Signal: XCPU (CPU limit exceeded)

SIGXFSZ = 25

Signal: XFSZ (File size limit exceeded)

SIG_BLOCK = 0

Sigprocmask flags: Add signals to the set of blocked signals.

SIG_DFL = 0

Signal handler: Default signal handler

SIG_ERR = -1

Signal handler: error

SIG_IGN = 1

Signal handler: Ignore signal

SIG_MAXSIG = `UnixType.SIG_MAXSIG`

Maximum system signal number.

SIG_SETMASK = 2

Sigprocmask flags: Set of blocked signals is given.

SIG_UNBLOCK = 1

Sigprocmask flags: Remove signals from the set set of blocked signals.

SI_PAD_SIZE = ((128 div sizeof (longint)) - 3)

Signal information pad size.

STAT_IFBLK = \$6000

File (`#rtl.baseunix.stat` (95) record) mode: Block device

STAT_IFCHR = \$2000

File (`#rtl.baseunix.stat` (95) record) mode: Character device

STAT_IFDIR = \$4000

File (#rtl.baseunix.stat (95) record) mode: Directory

STAT_IFIFO = \$1000

File (#rtl.baseunix.stat (95) record) mode: FIFO

STAT_IFLNK = \$a000

File (#rtl.baseunix.stat (95) record) mode: Link

STAT_IFMT = \$f000

File (#rtl.baseunix.stat (95) record) mode: File type bit mask

STAT_IFREG = \$8000

File (#rtl.baseunix.stat (95) record) mode: Regular file

STAT_IFSOCK = \$c000

File (#rtl.baseunix.stat (95) record) mode: Socket

STAT_IRGRP = STAT_IROTH shl 3

File (#rtl.baseunix.stat (95) record) mode: Group read permission

STAT_IROTH = \$4

File (#rtl.baseunix.stat (95) record) mode: Other read permission

STAT_IRUSR = STAT_IROTH shl 6

File (#rtl.baseunix.stat (95) record) mode: Owner read permission

STAT_IRWXG = STAT_IRWXO shl 3

File (#rtl.baseunix.stat (95) record) mode: Group permission bits mask

STAT_IRWXO = \$7

File (#rtl.baseunix.stat (95) record) mode: Other permission bits mask

STAT_IRWXU = STAT_IRWXO shl 6

File (#rtl.baseunix.stat (95) record) mode: Owner permission bits mask

STAT_ISGID = \$0400

File (#rtl.baseunix.stat (95) record) mode: GID bit set

STAT_ISUID = \$0800

File (#rtl.baseunix.stat (95) record) mode: UID bit set

STAT_ISVTX = \$0200

File (#rtl.baseunix.stat (95) record) mode: Sticky bit set

STAT_IWGRP = STAT_IWOTH shl 3

File (#rtl.baseunix.stat (95) record) mode: Group write permission

STAT_IWOTH = \$2

File (#rtl.baseunix.stat (95) record) mode: Other write permission

STAT_IWUSR = STAT_IWOTH shl 6

File (#rtl.baseunix.stat (95) record) mode: Owner write permission

STAT_IXGRP = STAT_IXOTH shl 3

File (#rtl.baseunix.stat (95) record) mode: Others execute permission

STAT_IXOTH = \$1

File (#rtl.baseunix.stat (95) record) mode: Others execute permission

STAT_IXUSR = STAT_IXOTH shl 6

File (#rtl.baseunix.stat (95) record) mode: Others execute permission

SYS_NMLN = UnixType.SYS_NMLN

Max system name length.

S_IFBLK = 24576

File (#rtl.baseunix.stat (95) record) mode: Block device

S_IFCHR = 8192

File (#rtl.baseunix.stat (95) record) mode: Character device

S_IFDIR = 16384

File (#rtl.baseunix.stat (95) record) mode: Directory

S_IFIFO = 4096

File (#rtl.baseunix.stat (95) record) mode: FIFO

S_IFLNK = 40960

File (#rtl.baseunix.stat (95) record) mode: Link

S_IFMT = 61440

File (#rtl.baseunix.stat (95) record) mode: File type bit mask

S_IFREG = 32768

File (#rtl.baseunix.stat (95) record) mode: Regular file

S_IFSOCK = 49152

File (#rtl.baseunix.stat (95) record) mode: Socket

Wait_Any = -1

#rtl.baseunix.fpWaitPID (147): Wait on any process

Wait_Clone = \$80000000

#rtl.baseunix.fpWaitPID (147): Wait on clone processes only.

Wait_MyPGRP = 0

#rtl.baseunix.fpWaitPID (147): Wait processes from current process group

Wait_NoHang = 1

#rtl.baseunix.fpWaitPID (147): Do not wait

Wait_UnTraced = 2

#rtl.baseunix.fpWaitPID (147): Also report stopped but untraced processes

Types

cInt = UnixType.cInt

C type: integer (natural size)

cInt16 = UnixType.cInt16

C type: 16 bits sized, signed integer.

cInt32 = UnixType.cInt32

C type: 32 bits sized, signed integer.

cInt64 = UnixType.cInt64

C type: 64 bits sized, signed integer.

cInt8 = UnixType.cInt8

C type: 8 bits sized, signed integer.

clock_t = UnixType.clock_t

Clock ticks type

cLong = UnixType.cLong

C type: long signed integer (double sized)

`cshort = UnixType.cshort`

C type: short signed integer (half sized)

`cuchar = UnixType.cuchar`

C type: unsigned character

`cUInt = UnixType.cUInt`

C type: unsigned integer (natural size)

`cUInt16 = UnixType.cUInt16`

C type: 16 bits sized, unsigned integer.

`cUInt32 = UnixType.cUInt32`

C type: 32 bits sized, unsigned integer.

`cUInt64 = UnixType.cUInt64`

C type: 64 bits sized, unsigned integer.

`cUInt8 = UnixType.cUInt8`

C type: 8 bits sized, unsigned integer.

`cuLong = UnixType.cuLong`

C type: long unsigned integer (double sized)

`cushort = UnixType.cushort`

C type: short unsigned integer (half sized)

`dev_t = UnixType.dev_t`

Device descriptor type.

`gid_t = UnixType.gid_t`

Group ID type.

`ino_t = UnixType.ino_t`

Inode type.

`mode_t = UnixType.mode_t`

Inode mode type.

`nlink_t = UnixType.nlink_t`

Number of links type.

`off_t = UnixType.off_t`

Offset type.

`pcInt = UnixType.pcInt`

Pointer to `cInt` (1244) type.

`pClock = UnixType.pClock`

Pointer to `TClock` (1249) type.

`pcLong = UnixType.pcLong`

Pointer to `cLong` (1245) type.

`pcshort = UnixType.pcsshort`

Pointer to `cShort` (1245) type.

`pcUInt = UnixType.pcUInt`

Pointer to `cUInt` (1245) type.

`pculong = UnixType.pculong`

Pointer to `cuLong` (1245) type.

`pcushort = UnixType.pcushort`

Pointer to `cuShort` (1245) type.

`pDev = UnixType.pDev`

Pointer to `TDev` (1249) type.

`pfpstate = ^tfpstate`

Pointer to `tfpstate` (1249) record.

`pGid = UnixType.pGid`

Pointer to `TGid` (1250) type.

`pid_t = UnixType.pid_t`

Process ID type.

`pIno = UnixType.pIno`

Pointer to `TIno` (1250) type.

`pMode = UnixType.pMode`

Pointer to TMode (1250) type.

`pnLink = UnixType.pnLink`

Pointer to TnLink (1250) type.

`pOff = UnixType.pOff`

Pointer to TOff (1250) type.

`pPid = UnixType.pPid`

Pointer to TPid (1250) type.

`PSigActionRec = ^SigActionRec`

Pointer to SigActionRec (1248) record type.

`PSigInfoRec = ^SigInfoRec`

Pointer to SigInfoRec (1248) record type.

`PSignalHandler = ^SignalHandler`

Pointer to SignalHandler (1248) type.

`PSignalRestorer = ^SignalRestorer`

Pointer to SignalRestorer (1249) type

`PSigSet = ^SigSet`

Pointer to SigSet (1249) type.

`psigset_t = PSigSet`

Pointer to sigset_t (1249) type.

`pSize = UnixType.pSize`

Pointer to TSize (1251) type.

`pSocklen = UnixType.pSocklen`

Pointer to TSockLen (1251) type.

`psSize = UnixType.psSize`

Pointer to TsSize (1251) type

`pthread_cond_t = UnixType.pthread_cond_t`

Thread conditional variable type.

`pthread_mutex_t = UnixType.pthread_mutex_t`

Thread mutex type.

```
pthread_t = UnixType.pthread_t
```

Posix thread type.

```
pTime = UnixType.pTime
```

Pointer to TTime (1251) type.

```
ptimespec = UnixType.ptimespec
```

Pointer to timespec (1250) type.

```
ptimeval = UnixType.ptimeval
```

Pointer to timeval (1250) type.

```
ptimezone = ^timezone
```

Pointer to TimeZone (1250) record.

```
ptime_t = UnixType.ptime_t
```

Pointer to time_t (1250) type.

```
pUId = UnixType.pUId
```

Pointer to TUid (1251) type.

```
SigActionRec = packed record
  sa_handler : SignalHandler;
  Sa_Mask : SigSet;
  Sa_Flags : LongInt;
  Sa_restorer : SignalRestorer;
end
```

Record used in #rtl.baseunix.fpSigAction (134) call.

```
SigInfoRec = record
  si_signo : LongInt;
  si_errno : LongInt;
  si_code : LongInt;
end
```

Record containing signal information.

```
SignalHandler = procedure(Sig: LongInt)
```

Simple signal handler prototype

```
SignalRestorer = procedure
```

Signal restorer function prototype

```
SigSet = Array[0..wordsinsigset-1] of cInt
```

Signal set type

```
sigset_t = SigSet
```

Signal set type

```
size_t = UnixType.size_t
```

Size specification type.

```
socklen_t = UnixType.socklen_t
```

Socket address length type.

```
ssize_t = UnixType.ssize_t
```

Small size type.

```
TClock = UnixType.TClock
```

Alias for clock_t (1244) type.

```
TDev = UnixType.TDev
```

Alias for dev_t (1245) type.

```
tfpreg = record
  significand : Array[0..3] of Word;
  exponent   : Word;
end
```

Record describing floating point register in signal handler.

```
tfpstate = record
  cw : cardinal;
  sw : cardinal;
  tag : cardinal;
  ipoff : cardinal;
  cssel : cardinal;
  dataoff : cardinal;
  datasel : cardinal;
  st : Array[0..7] of tfpreg;
  status : cardinal;
end
```

Record describing floating point unit in signal handler.

```
TFSearchOption = (NoCurrentDirectory, CurrentDirectoryFirst,
                  CurrentDirectoryLast)
```

Table 32.2: Enumeration values for type TFSearchOption

Value	Explanation
CurrentDirectoryFirst	Search the current directory first, before all directories in the search path.
CurrentDirectoryLast	Search the current directory last, after all directories in the search path
NoCurrentDirectory	Do not search the current directory unless it is specified in the search path.

Describes the search strategy used by FSearch ([1261](#))

TGid = UnixType.TGid

Alias for gid_t ([1245](#)) type.

timespec = UnixType.timespec

Short time specification type.

timeval = UnixType.timeval

Time specification type.

```
timezone = packed record
  tz_minuteswest : cInt;
  tz_dsttime : cInt;
end
```

Record describing a timezone

time_t = UnixType.time_t

Time span type

TIno = UnixType.TIno

Alias for ino_t ([1245](#)) type.

TMode = UnixType.TMode

Alias for mode_t ([1245](#)) type.

TnLink = UnixType.TnLink

Alias for nlink_t ([1246](#)) type.

TOff = UnixType.TOff

Alias for off_t ([1246](#)) type.

TPid = UnixType.TPid

Alias for pid_t ([1246](#)) type.

`Tpipe = baseunix.tfiles`

Array describing a pipe pair of filedescriptors.

`TSigAction = procedure(Sig: LongInt; SigContext: SigContextRec)`

Extended signal handler prototype.

`TSigActionRec = SigActionRec`

Alias for `SigActionRec` (1248) record type.

`TSigSet = SigSet`

Alias for `SigSet` (1249) type.

`TSize = UnixType.TSize`

Alias for `size_t` (1249) type

`TSocklen = UnixType.TSocklen`

Alias for `socklen_t` (1249) type.

`TsSize = UnixType.TsSize`

Alias for `ssize_t` (1249) type

`tstatfs = UnixType.TStatFs`

`TTime = UnixType.TTime`

Alias for `TTime` (1251) type.

`Ttimespec = UnixType.Ttimespec`

Alias for `TimeSpec` (1250) type.

`TTimeVal = UnixType.TTimeVal`

Alias for `timeval` (1250) type.

`TTimeZone = timezone`

Alias for `TimeZone` (1250) record.

`TUid = UnixType.TUid`

Alias for `uid_t` (1251) type.

`uid_t = UnixType.uid_t`

User ID type

Variables

`tzdaylight` : Boolean

Indicates whether daylight savings time is active.

`tzname` : Array[boolean] of pchar

Timezone name.

`Tzseconds` : LongInt = 0

32.3 Procedures and functions

AssignPipe

Synopsis: Create a set of pipe file handlers

Declaration: `function AssignPipe(var pipe_in: cInt; var pipe_out: cInt) : cInt`
`function AssignPipe(var pipe_in: text; var pipe_out: text) : cInt`
`function AssignPipe(var pipe_in: file; var pipe_out: file) : cInt`

Visibility: default

Description: AssignPipe creates a pipe, i.e. two file objects, one for input, one for output. What is written to Pipe_out, can be read from Pipe_in.

This call is overloaded. The in and out pipe can take three forms: an typed or untyped file, a text file or a file descriptor.

If a text file is passed then reading and writing from/to the pipe can be done through the usual `Readln(Pipe_in, ...)` and `Writeln(Pipe_out, ...)` procedures.

The function returns `True` if everything went successfully, `False` otherwise.

Errors: In case the function fails and returns `False`, extended error information is returned by the `FpGetErrno` (112) function:

sys_enfile Too many file descriptors for this process.

sys_enfile The system file table is full.

See also: `POpen` (1265), `#rtl.baseunix.FpMkFifo` (120)

Listing: ./unixex/ex36.pp

Program Example36;

{ Program to demonstrate the AssignPipe function. }

Uses BaseUnix, Unix;

Var pipi, pipo : Text;
s : **String**;

begin

Writeln ('Assigning Pipes. ');

If assignpipe(pipi, pipo) <> 0 **then**

Writeln ('Error assigning pipes ! ', fpgeterrno);

```

Writeln ( 'Writing to pipe, and flushing.' );
Writeln ( pipo, 'This is a textstring' ); close ( pipo );
Writeln ( 'Reading from pipe.' );
While not eof ( pipi ) do
begin
  Readln ( pipi, s );
  Writeln ( 'Read from pipe : ', s );
end;
close ( pipi );
writeln ( 'Closed pipes.' );
writeln
end.

```

AssignStream

Synopsis: Assign stream for in and output to a program

Declaration: function AssignStream (var StreamIn: text; var Streamout: text;
 const Prog: ansiString;
 const args: Array[] of ansistring) : cInt
 function AssignStream (var StreamIn: text; var Streamout: text;
 var streamerr: text; const Prog: ansiString;
 const args: Array[] of ansistring) : cInt

Visibility: default

Description: AssignStream creates a 2 or 3 pipes, i.e. two (or three) file objects, one for input, one for output, (and one for standard error) the other ends of these pipes are connected to standard input and output (and standard error) of Prog. Prog is the name of a program (including path) with options, which will be executed.

What is written to StreamOut, will go to the standard input of Prog. Whatever is written by Prog to its standard output can be read from StreamIn. Whatever is written by Prog to its standard error read from StreamErr, if present.

Reading and writing happens through the usual Readln (StreamIn, ...) and Writeln (StreamOut, ...) procedures.

Remark: You should *not* use Reset or Rewrite on a file opened with POpen. This will close the file before re-opening it again, thereby closing the connection with the program.

The function returns the process ID of the spawned process, or -1 in case of error.

Errors: Extended error information is returned by the FpGetErrno ([112](#)) function.

sys_emfile Too many file descriptors for this process.

sys_emfile The system file table is full.

Other errors include the ones by the fork and exec programs

See also: AssignPipe ([1252](#)), POpen ([1265](#))

Listing: ./unixex/ex38.pp

Program Example38;

{ Program to demonstrate the AssignStream function. }

Uses BaseUnix, Unix;

```

Var Si,So : Text;
      S : String;
      i : longint;

begin
  if not (paramstr(1)='-son') then
    begin
      Writeln ('Calling son');
      Assignstream (Si,So,'./ex38 -son');
      if fpgeterrno <> 0 then
        begin
          writeln ('AssignStream failed !');
          halt(1);
        end;
      Writeln ('Speaking to son');
      For i:=1 to 10 do
        begin
          writeln (so,'Hello son !');
          if ioreult <> 0 then writeln ('Can't speak to son...');
        end;
      For i:=1 to 3 do writeln (so,'Hello chap !');
      close (so);
      while not eof(si) do
        begin
          readln (si,s);
          writeln ('Father: Son said : ',S);
        end;
      Writeln ('Stopped conversation');
      Close (Si);
      Writeln ('Put down phone');
    end
  Else
    begin
      Writeln ('This is the son ');
      While not eof (input) do
        begin
          readln (s);
          if pos ('Hello son !',S) <> 0 then
            Writeln ('Hello Dad !')
          else
            writeln ('Who are you ?');
          end;
        close (output);
      end
    end.

```

FpExecL

Synopsis: Execute process (using argument list, environment)

Declaration: `function FpExecL(const PathName: AnsiString;
const S: Array[] of AnsiString) : cInt`

Visibility: default

Description: FpExecL replaces the currently running program with the program, specified in PathName. S is

an array of command options. The executable in `PathName` must be an absolute pathname. The current process' environment is passed to the program. On success, `FpExecL` does not return.

Errors: Extended error information is returned by the `FpGetErrno` (112) function:

sys_eaccessFile is not a regular file, or has no execute permission. A component of the path has no search permission.

sys_epermThe file system is mounted *noexec*.

sys_e2bigArgument list too big.

sys_enoexecThe magic number in the file is incorrect.

sys_enoentThe file does not exist.

sys_enomemNot enough memory for kernel, or to split command line.

sys_enotdirA component of the path is not a directory.

sys_eloopThe path contains a circular reference (via symlinks).

See also: `#rtl.baseunix.fpexecve` (106), `FpExecv` (1257), `FpExecvp` (1258), `FpExecle` (1255), `FpExeclp` (1256), `#rtl.baseunix.FpFork` (109)

Listing: `./unixex/ex77.pp`

Program `Example77`;

{ Program to demonstrate the FpExecL function. }

Uses `Unix`, `strings`;

begin

{ Execute 'ls -l', with current environment. }

{ 'ls' is NOT looked for in PATH environment variable. }

`FpExecL ('/bin/ls' , ['-l']);`

end.

FpExecLE

Synopsis: Execute process (using argument list, environment)

Declaration: `function FpExecLE(const PathName: AnsiString;
const S: Array[] of AnsiString; MyEnv: ppchar) : cInt`

Visibility: default

Description: `FpExecLE` replaces the currently running program with the program, specified in `PathName`. `S` is an array of command options. The executable in `PathName` must be an absolute pathname. The environment in `MyEnv` is passed to the program. On success, `FpExecLE` does not return.

Errors: Extended error information is returned by the `FpGetErrno` (112) function:

sys_eaccessFile is not a regular file, or has no execute permission. A component of the path has no search permission.

sys_epermThe file system is mounted *noexec*.

sys_e2bigArgument list too big.

sys_enoexecThe magic number in the file is incorrect.

sys_enoentThe file does not exist.

sys_enomemNot enough memory for kernel, or to split command line.

sys_enotdirA component of the path is not a directory.

sys_eloopThe path contains a circular reference (via symlinks).

See also: #rtl.baseunix.fpexecve (106), FpExecv (1257), FpExecvp (1258), FpExecl (1254), FpExeclp (1256), #rtl.baseunix.FpFork (109)

Listing: ./unixex/ex11.pp

Program Example11;

{ Program to demonstrate the Execl function. }

Uses Unix, strings;

begin

{ Execute 'ls -l', with current environment. }
{ 'ls' is NOT looked for in PATH environment variable. }
{ envp is defined in the system unit. }
 Execl ('/bin/ls -l', envp);

end.

FpExecLP

Synopsis: Execute process (using argument list, environment; search path)

Declaration: function FpExecLP(const PathName: AnsiString;
 const S: Array[] of AnsiString) : cInt

Visibility: default

Description: FpExecLP replaces the currently running program with the program, specified in PathName. S is an array of command options. The executable in PathName is searched in the path, if it isn't an absolute filename. The current environment is passed to the program. On success, FpExecLP does not return.

Errors: Extended error information is returned by the FpGetErrno (112) function:

sys_eaccessFile is not a regular file, or has no execute permission. A component of the path has no search permission.

sys_epermThe file system is mounted *noexec*.

sys_e2bigArgument list too big.

sys_enoexecThe magic number in the file is incorrect.

sys_enoentThe file does not exist.

sys_enomemNot enough memory for kernel, or to split command line.

sys_enotdirA component of the path is not a directory.

sys_eloopThe path contains a circular reference (via symlinks).

See also: #rtl.baseunix.fpexecve (106), FpExecv (1257), FpExecvp (1258), FpExecl (1255), FpExecl (1254), #rtl.baseunix.FpFork (109)

Listing: ./unixex/ex76.pp

Program Example76;

{ Program to demonstrate the FpExeclp function. }

Uses Unix , strings ;

begin

{ Execute 'ls -l', with current environment. }
{ 'ls' is looked for in PATH environment variable. }
{ envp is defined in the system unit. }
 FpExeclp ('ls' , ['-l']);

end.

FpExecV

Synopsis: Execute process

Declaration: function FpExecV(const PathName: AnsiString; args: ppchar) : cInt

Visibility: default

Description: FpExecV replaces the currently running program with the program, specified in PathName. It gives the program the options in args. This is a pointer to an array of pointers to null-terminated strings. The last pointer in this array should be nil. The current environment is passed to the program. On success, FpExecV does not return.

Errors: Extended error information is returned by the FpGetErrno ([112](#)) function:

sys_eaccessFile is not a regular file, or has no execute permission. A component of the path has no search permission.

sys_epermThe file system is mounted *noexec*.

sys_e2bigArgument list too big.

sys_enoexecThe magic number in the file is incorrect.

sys_enoentThe file does not exist.

sys_enomemNot enough memory for kernel.

sys_enotdirA component of the path is not a directory.

sys_eloopThe path contains a circular reference (via symlinks).

See also: #rtl.baseunix.fpexecve ([106](#)), FpExecvp ([1258](#)), FpExecl ([1255](#)), FpExecl ([1254](#)), FpExeclp ([1256](#)), #rtl.baseunix.FpFork ([109](#))

Listing: ./unixex/ex8.pp

Program Example8;

{ Program to demonstrate the Execv function. }

Uses Unix , strings ;

Const Arg0 : PChar = '/bin/ls';
 Arg1 : Pchar = '-l';

Var PP : PPchar;

```

begin
  GetMem (PP, 3*SizeOf(Pchar));
  PP[0] := Arg0;
  PP[1] := Arg1;
  PP[3] := Nil;
  { Execute '/bin/ls -l', with current environment }
  fpExecv ('/bin/ls', pp);
end.

```

FpExecVP

Synopsis: Execute process, search path

Declaration: function FpExecVP(const PathName: AnsiString; args: ppchar) : cInt

Visibility: default

Description: FpExecVP replaces the currently running program with the program, specified in PathName. The executable in path is searched in the path, if it isn't an absolute filename. It gives the program the options in args. This is a pointer to an array of pointers to null-terminated strings. The last pointer in this array should be nil. The current environment is passed to the program. On success, `execvp` does not return.

Errors: Extended error information is returned by the FpGetErrno (112) function:

sys_eaccessFile is not a regular file, or has no execute permission. A component of the path has no search permission.

sys_epermThe file system is mounted *noexec*.

sys_e2bigArgument list too big.

sys_enoexecThe magic number in the file is incorrect.

sys_enoentThe file does not exist.

sys_enomemNot enough memory for kernel.

sys_enotdirA component of the path is not a directory.

sys_eloopThe path contains a circular reference (via symlinks).

See also: #rtl.baseunix.fplexecve (106), FpExecv (1257), FpExecl (1255), FpExecl (1254), FpExeclp (1256), #rtl.baseunix.FpFork (109)

Listing: ./unixex/ex79.pp

Program Example79;

{ Program to demonstrate the FpExecVP function. }

Uses Unix, strings;

Const Arg0 : PChar = 'ls';
 Arg1 : PChar = '-l';

Var PP : PPchar;

```

begin
  GetMem (PP, 3 * SizeOf (Pchar));
  PP[0] := Arg0;
  PP[1] := Arg1;
  PP[2] := Nil;
  { Execute 'ls -l', with current environment. }
  { 'ls' is looked for in PATH environment variable. }
  fpExecvp ('ls', pp);
end.

```

FpExecVPE

Synopsis: Execute process, search path using environment

Declaration: `function FpExecVPE(const PathName: AnsiString; args: ppchar; env: ppchar) : cInt`

Visibility: default

Description: `FpExecVP` replaces the currently running program with the program, specified in `PathName`. The executable in path is searched in the path, if it isn't an absolute filename. It gives the program the options in `args`. This is a pointer to an array of pointers to null-terminated strings. The last pointer in this array should be nil. The environment in `Env` is passed to the program. On success, `execvp` does not return.

Errors: Extended error information is returned by the `FpGetErrno` (112) function:

sys_eaccessFile is not a regular file, or has no execute permission. A component of the path has no search permission.

sys_epermThe file system is mounted *noexec*.

sys_e2bigArgument list too big.

sys_enoexecThe magic number in the file is incorrect.

sys_enoentThe file does not exist.

sys_enomemNot enough memory for kernel.

sys_enotdirA component of the path is not a directory.

sys_eloopThe path contains a circular reference (via symlinks).

See also: `#rtl.baseunix.fplexecve` (106), `FpExecv` (1257), `FpExecl` (1255), `FpExecl` (1254), `FpExeclp` (1256), `#rtl.baseunix.FpFork` (109)

Listing: `./unixex/ex79.pp`

Program Example79;

{ Program to demonstrate the FpExecVP function. }

Uses Unix, strings;

Const Arg0 : PChar = 'ls';
 Arg1 : PChar = '-l';

Var PP : PPchar;

```

begin
  GetMem (PP,3*SizeOf(Pchar));
  PP[0]:=Arg0;
  PP[1]:=Arg1;
  PP[2]:=Nil;
  { Execute 'ls -l', with current environment. }
  { 'ls' is looked for in PATH environment variable. }
  fpExecvp ('ls',pp);
end.

```

fpFlock

Synopsis: Lock a file (advisory lock)

Declaration: `function fpFlock(fd: cInt;mode: cInt) : cInt`
`function fpFlock(var T: text;mode: cInt) : cInt`
`function fpFlock(var F: File;mode: cInt) : cInt`

Visibility: default

Description: `FpFlock` implements file locking. it sets or removes a lock on the file `F`. `F` can be of type `Text` or `File`, or it can be a linux filedescriptor (a longint) `Mode` can be one of the following constants :

LOCK_SHsets a shared lock.

LOCK_EXsets an exclusive lock.

LOCK_UNunlocks the file.

LOCK_NBThis can be OR-ed together with the other. If this is done the application doesn't block when locking.

The function returns zero if successful, a nonzero return value indicates an error.

Errors: Extended error information is returned by the `FpGetErrno` ([112](#)) function:

See also: `#rtl.baseunix.FpFcntl` ([108](#)), `FSync` ([1263](#))

fpgettimeofday

Synopsis: Return kernel time of day in GMT

Declaration: `function fpgettimeofday(tp: ptimeval;tzp: ptimezone) : cInt`

Visibility: default

Description: `FpGetTimeOfDay` returns the number of seconds since 00:00, January 1 1970, GMT in a `timeval` record. This time NOT corrected any way, not taking into account timezones, daylight savings time and so on.

It is simply a wrapper to the kernel system call.

Errors: None.

fpSystem

Synopsis: Execute and feed command to system shell

Declaration: `function fpSystem(const Command: AnsiString) : cInt`

Visibility: default

Description: Shell invokes the bash shell (/bin/sh), and feeds it the command Command (using the -c option). The function then waits for the command to complete, and then returns the exit status of the command, or 127 if it could not complete the FpFork (109) or FpExecve (106) calls.

Errors: Errors are reported in LinuxError.

See also: POpen (1265), Shell (1267), #rtl.baseunix.FpFork (109), #rtl.baseunix.fpexecve (106)

Listing: ./unixex/ex80.pp

```

program example56;

uses Unix;

{ Program to demonstrate the Shell function }

Var S : Longint;

begin
  WriteLn ( 'Output of ls -l *.pp' );
  S:=fpSystem( 'ls -l *.pp' );
  WriteLn ( 'Command exited with status : ',S);
end.

```

FSearch

Synopsis: Search for file in search path.

Declaration: `function FSearch(const path: AnsiString;dirlist: Ansistring;
CurrentDirStrategy: TFSearchOption) : AnsiString
function FSearch(const path: AnsiString;dirlist: AnsiString)
: AnsiString`

Visibility: default

Description: FSearch searches in DirList, a colon separated list of directories, for a file named Path. It then returns a path to the found file.

The CurrentDirStrategy determines how the current directory is treated when searching:

NoCurrentDirectory Do not search the current directory unless it is specified in the search path.

CurrentDirectoryFirst Search the current directory first, before all directories in the search path.

CurrentDirectoryLast Search the current directory last, after all directories in the search path

It is mainly provided to mimic DOS search path behaviour. Default behaviour is to search the current directory first.

Errors: An empty string if no such file was found.

See also: #rtl.unixutil.FNMatch (1285)

Listing: ./unixex/ex46.pp

Program Example46;

{ Program to demonstrate the FSearch function. }

Uses BaseUnix, Unix, Strings;

begin

WriteIn ('ls is in : ', FSearch ('ls', **strpas**(fpGetenv('PATH'))));
end.

fStatFS

Synopsis: Retrieve filesystem information from a file descriptor.

Declaration: `function fStatFS(Fd: cInt; var Info: tstatfs) : cInt`

Visibility: default

Description: `fStatFS` returns in `Info` information about the filesystem on which the file with file descriptor `fd` resides. `Info` is of type `TStatFS` ([1280](#)).

The function returns zero if the call was succesful, a nonzero value is returned if the call failed.

Errors: Extended error information is returned by the `FpGetErrno` ([112](#)) function:

sys_enotdirA component of `Path` is not a directory.

sys_einvalInvalid character in `Path`.

sys_enoent`Path` does not exist.

sys_eaccessSearch permission is denied for component in `Path`.

sys_eloopA circular symbolic link was encountered in `Path`.

sys_eioAn error occurred while reading from the filesystem.

See also: `StatFS` ([1269](#)), `#rtl.baseunix.FpLStat` ([119](#))

Listing: ./unixex/ex91.pp

program Example30;

{ Program to demonstrate the FSStat function. }

uses BaseUnix, Unix, UnixType;

var s : **string**;

 fd : cInt;

 info : tstatfs;

begin

writeln ('Info about current partition : ');

 s := ' . ';

while s <> 'q' **do**

begin

 Fd := fpOpen(S, O_RDONLY);

if (fd >= 0) **then**

begin

```

    if fstatfs (fd,info)<>0 then
        begin
            writeln('Fstat failed. Errno : ',fpgeterrno);
            halt (1);
        end;
    FpClose(fd);
    writeln;
    writeln ('Result of fsstat on file ''',s, ''.');
    writeln ('fstype   : ',info.fstype);
    writeln ('bsize    : ',info.bsize);
    writeln ('bfree     : ',info.bfree);
    writeln ('bavail    : ',info.bavail);
    writeln ('files     : ',info.files);
    writeln ('ffree     : ',info.ffree);
    {$ifdef FreeBSD}
    writeln ('fsid      : ',info.fsid[0]);
    {$else}
    writeln ('fsid      : ',info.fsid[0]);
    writeln ('Namelen   : ',info.namelen);
    {$endif}
    write ('Type name of file to do fsstat. (q quits) : ');
    readln (s)

end;
end;
end.
```

fsync

Synopsis: Synchronize file's kernel data with disk.

Declaration: `function fsync(fd: cInt) : cInt`

Visibility: default

Description: `Fsync` synchronizes the kernel data for file `fd` (the cache) with the disk. The call will not return till all file data was written to disk.

If the call was succesfull, 0 is returned. On failure, a nonzero value is returned.

Errors: Extended error information is returned by the `FpGetErrno` ([112](#)) function:

See also: `FpFLock` ([1260](#))

GetDomainName

Synopsis: Return current domain name

Declaration: `function GetDomainName : String`

Visibility: default

Description: Get the domain name of the machine on which the process is running. An empty string is returned if the domain is not set.

Errors: None.

See also: `GetHostName` ([1264](#))

Listing: ./unixex/ex39.pp

Program Example39;

{ Program to demonstrate the GetDomainName function. }

Uses Unix;

begin

WriteLn ('Domain name of this machine is : ',GetDomainName);
end.

GetHostName

Synopsis: Return host name

Declaration: `function GetHostName : String`

Visibility: default

Description: Get the hostname of the machine on which the process is running. An empty string is returned if hostname is not set.

Errors: None.

See also: `GetDomainName` ([1263](#))

Listing: ./unixex/ex40.pp

Program Example40;

{ Program to demonstrate the GetHostName function. }

Uses unix;

begin

WriteLn ('Name of this machine is : ',GetHostName);
end.

GetLocalTimezone

Synopsis: Return local timzeone information

Declaration: `procedure GetLocalTimezone(timer: cInt;var leap_correct: cInt;
 var leap_hit: cInt)
 procedure GetLocalTimezone(timer: cInt)`

Visibility: default

Description: `GetLocalTimezone` returns the local timezone information. It also initializes the `TZSeconds` variable, which is used to correct the epoch time to local time.

There should never be any need to call this function directly. It is called by the initialization routines of the Linux unit.

See also: `GetTimezoneFile` ([1265](#)), `ReadTimezoneFile` ([1266](#))

GetTimezoneFile

Synopsis: Return name of timezone information file

Declaration: `function GetTimezoneFile : String`

Visibility: default

Description: `GetTimezoneFile` returns the location of the current timezone file. The location of file is determined as follows:

- 1.If `/etc/timezone` exists, it is read, and the contents of this file is returned. This should work on Debian systems.
- 2.If `/usr/lib/zoneinfo/localtime` exists, then it is returned. (this file is a symlink to the timezone file on SuSE systems)
- 3.If `/etc/localtime` exists, then it is returned. (this file is a symlink to the timezone file on RedHat systems)

Errors: If no file was found, an empty string is returned.

See also: `ReadTimezoneFile` ([1266](#))

PClose

Synopsis: Close file opened with `POpen` ([1265](#))

Declaration: `function PClose(var F: file) : cInt`
`function PClose(var F: text) : cInt`

Visibility: default

Description: `PClose` closes a file opened with `POpen` ([1265](#)). It waits for the command to complete, and then returns the exit status of the command.

For an example, see `POpen` ([1265](#))

Errors: Extended error information is returned by the `FpGetErrno` ([112](#)) function.

See also: `POpen` ([1265](#))

POpen

Synopsis: Pipe file to standard input/output of program

Declaration: `function POpen(var F: text;const Prog: String;rw: Char) : cInt`
`function POpen(var F: file;const Prog: String;rw: Char) : cInt`

Visibility: default

Description: `POpen` runs the command specified in `Prog`, and redirects the standard in or output of the command to the other end of the pipe `F`. The parameter `rw` indicates the direction of the pipe. If it is set to 'W', then `F` can be used to write data, which will then be read by the command from `stdin`. If it is set to 'R', then the standard output of the command can be read from `F`. `F` should be reset or rewritten prior to using it. `F` can be of type `Text` or `File`. A file opened with `POpen` can be closed with `Close`, but also with `PClose` ([1265](#)). The result is the same, but `PClose` returns the exit status of the command `Prog`.

Errors: Extended error information is returned by the `FpGetErrno` ([112](#)) function. Errors are essentially those of the `Execve`, `Dup` and `AssignPipe` commands.

See also: [AssignPipe \(1252\)](#), [PClose \(1265\)](#)

Listing: ./unixex/ex37.pp

Program Example37;

```
{ Program to demonstrate the Popen function . }
```

```
uses BaseUnix, Unix;

var f : text;
    i : longint;

begin
  writeln ( 'Creating a shell script to which echoes its arguments' );
  writeln ( 'and input back to stdout' );
  assign ( f, 'test21a' );
  rewrite ( f );
  writeln ( f, '#!/bin/sh' );
  writeln ( f, 'echo this is the child speaking.... ' );
  writeln ( f, 'echo got arguments \*"${*}"\*' );
  writeln ( f, 'cat' );
  writeln ( f, 'exit 2' );
  writeln ( f );
  close ( f );
  fpchmod ( 'test21a', &755 );
  popen ( f, './test21a arg1 arg2 ', 'W' );
  if fpgeterrno <> 0 then
    writeln ( 'error from POpen : errno : ', fpgeterrno );
  for i:=1 to 10 do
    writeln ( f, 'This is written to the pipe, and should appear on stdout.' );
  Flush(f);
  Writeln ( 'The script exited with status : ', PClose ( f ) );
  writeln;
  writeln ( 'Press <return> to remove shell script.' );
  readln;
  assign ( f, 'test21a' );
  erase ( f );
end.
```

ReadTimezoneFile

Synopsis: Read the timezone file and initialize time routines

Declaration: procedure ReadTimezoneFile(fn: String)

Visibility: default

Description: ReadTimezoneFile reads the timezone file fn and initializes the local time routines based on the information found there.

There should be no need to call this function. The initialization routines of the linux unit call this routine at unit startup.

Errors: None.

See also: [GetTimezoneFile \(1265\)](#), [GetLocalTimezone \(1264\)](#)

SeekDir

Synopsis: Seek to position in directory

Declaration: `procedure SeekDir(p: pDir; loc: cLong)`

Visibility: default

Description: `SeekDir` sets the directory pointer to the `loc`-th entry in the directory structure pointed to by `p`.

For an example, see `#rtl.baseunix.fpOpenDir` (125).

Errors: Extended error information is returned by the `FpGetErrno` (112) function:

See also: `#rtl.baseunix.fpCloseDir` (103), `#rtl.baseunix.fpReadDir` (129), `#rtl.baseunix.fpOpenDir` (125), `TellDir` (1270)

SelectText

Synopsis: Wait for event on text file.

Declaration: `function SelectText(var T: Text; TimeOut: ptimeval) : cInt`
`function SelectText(var T: Text; TimeOut: cInt) : cInt`

Visibility: default

Description: `SelectText` executes the `FpSelect` (131) call on a file of type `Text`. You can specify a timeout in `TimeOut`. The `SelectText` call determines itself whether it should check for read or write, depending on how the file was opened : With `Reset` it is checked for reading, with `Rewrite` and `Append` it is checked for writing.

Errors: See `#rtl.baseunix.FpSelect` (131). `SYS_EBADF` can also mean that the file wasn't opened.

See also: `#rtl.baseunix.FpSelect` (131)

Shell

Synopsis: Execute and feed command to system shell

Declaration: `function Shell(const Command: String) : cInt`
`function Shell(const Command: AnsiString) : cInt`

Visibility: default

Description: `Shell` invokes the bash shell (`/bin/sh`), and feeds it the command `Command` (using the `-c` option). The function then waits for the command to complete, and then returns the exit status of the command, or 127 if it could not complete the `FpFork` (109) or `FpExecve` (106) calls.

Errors: Extended error information is returned by the `FpGetErrno` (112) function:

See also: `POpen` (1265), `FpSystem` (1261), `#rtl.baseunix.FpFork` (109), `#rtl.baseunix.fpexecve` (106)

Listing: `./unixex/ex56.pp`

program example56;

uses Unix;

{ Program to demonstrate the Shell function }

```

Var S : Longint;

begin
  WriteLn ( 'Output of ls -l *.pp' );
  S:= Shell ( 'ls -l *.pp' );
  WriteLn ( 'Command exited wwith status : ',S);
end.

```

SigRaise

Synopsis: Raise a signal (send to current process)

Declaration: `procedure SigRaise(sig: Integer)`

Visibility: default

Description: `SigRaise` sends a `Sig` signal to the current process.

Errors: None.

See also: `#rtl.baseunix.FpKill` ([116](#)), `#rtl.baseunix.FpGetPid` ([114](#))

Listing: `./unixex/ex65.pp`

Program `example64`;

{ Program to demonstrate the SigRaise function. }

uses `Unix, BaseUnix`;

Var

`oa, na : PSigActionRec`;

Procedure `DoSig(sig : Longint); cdecl`;

begin

`writeln('Receiving signal: ', sig);`

end;

begin

`new(na);`

`new(oa);`

`na^.sa_handler:= TSigaction (@DoSig);`

`fillchar (na^.Sa_Mask, sizeof(na^.Sa_Mask), #0);`

`na^.Sa_Flags:=0;`

{ \$ifdef Linux }

// this member is linux only, and afaik even there arcane

`na^.Sa_Restorer:= Nil;`

{ \$endif }

if `fpSigAction (SigUsr1, na, oa) <> 0` **then**

begin

`writeln('Error: ', fpgeterrno);`

`halt(1);`

end;

`WriteLn('Sending USR1 (', sigusr1, ') signal to self.');`

`SigRaise(sigusr1);`

end.

StatFS

Synopsis: Retrieve filesystem information from a path.

Declaration: `function StatFS(Path: pchar;var Info: tstatfs) : cInt`
`function StatFS(Path: ansistring;var Info: tstatfs) : cInt`

Visibility: default

Description: StatFS returns in Info information about the filesystem on which the file Path resides. Info is of type TStatFS (1280).

The function returns zero if the call was succesful, a nonzero value is returned if the call failed.

Errors: Extended error information is returned by the FpGetErrno (112) function:

sys_enotdirA component of Path is not a directory.
sys_einvalInvalid character in Path.
sys_enoentPath does not exist.
sys_eaccessSearch permission is denied for component inPath.
sys_eloopA circular symbolic link was encountered in Path.
sys_eioAn error occurred while reading from the filesystem.

See also: #rtl.baseunix.FpStat (139), #rtl.baseunix.FpLStat (119)

Listing: ./unixex/ex91.pp

```

program Example30;

{ Program to demonstrate the FSStat function. }

uses BaseUnix, Unix, UnixType;

var s : string;
    fd : cint;
    info : tstatfs;

begin
  writeln ('Info about current partition : ');
  s:= '.';
  while s<>'q' do
    begin
      Fd:=fpOpen(S,O_RDONLY);
      if (fd>=0) then
        begin
          if fstatfs (fd,info)<>0 then
            begin
              writeln ('Fstat failed. Errno : ',fpgeterrno);
              halt (1);
            end;
          FpClose(fd);
          writeln;
          writeln ('Result of fsstat on file ',s,'');
          writeln ('fstype   : ',info.fstype);
          writeln ('bsize    : ',info.bsize);
          writeln ('bfree    : ',info.bfree);
          writeln ('bavail   : ',info.bavail);
          writeln ('files    : ',info.files);
        end;
    end;
  end;

```

```

    writeln ( 'ffree   : ', info. ffree );
    { $ifdef FreeBSD }
    writeln ( 'fsid    : ', info. fsid [0]);
    { $else }
    writeln ( 'fsid    : ', info. fsid [0]);
    writeln ( 'Namelen : ', info. namelen);
    { $endif }
    write ( 'Type name of file to do fsstat. (q quits) : ');
    readln ( s)

    end;
end;
end.

```

TellDir

Synopsis: Return current location in a directory

Declaration: function TellDir(p: pDir) : cLong

Visibility: default

Description: TellDir returns the current location in the directory structure pointed to by p. It returns -1 on failure.

For an example, see #rtl.baseunix.fpOpenDir ([125](#)).

Errors:

See also: #rtl.baseunix.fpCloseDir ([103](#)), #rtl.baseunix.fpReadDir ([129](#)), #rtl.baseunix.fpOpenDir ([125](#)), SeekDir ([1267](#))

WaitProcess

Synopsis: Wait for process to terminate.

Declaration: function WaitProcess(Pid: cInt) : cInt

Visibility: default

Description: WaitProcess waits for process PID to exit. WaitProcess is equivalent to the #rtl.baseunix.FpWaitPID ([147](#)) call:

```
FpWaitPid(PID, @result, 0)
```

Handles of Signal interrupts (errno=EINTR), and returns the Exitcode of Process PID (>=0) or - Status if it was terminated

Errors: None.

See also: #rtl.baseunix.FpWaitPID ([147](#)), #rtl.baseunix.WTERMSIG ([149](#)), #rtl.baseunix.WSTOPSIG ([149](#)), #rtl.baseunix.WIFEXITED ([148](#)), WIFSTOPPED ([1271](#)), #rtl.baseunix.WIFSIGNALED ([148](#)), W_EXITCODE ([1271](#)), W_STOPCODE ([1271](#)), #rtl.baseunix.WEXITSTATUS ([148](#))

WIFSTOPPED

Synopsis: Check whether the process is currently stopped.

Declaration: `function WIFSTOPPED(Status: Integer) : Boolean`

Visibility: default

Description: `WIFSTOPPED` checks `Status` and returns `true` if the process is currently stopped. This is only possible if `WUNTRACED` was specified in the options of `FpWaitPID` (147).

See also: `#rtl.baseunix.FpWaitPID` (147), `WaitProcess` (1270), `#rtl.baseunix.WTERMSIG` (149), `#rtl.baseunix.WSTOPSIG` (149), `#rtl.baseunix.WIFEXITED` (148), `#rtl.baseunix.WIFSIGNALLED` (148), `W_EXITCODE` (1271), `W_STOPCODE` (1271), `#rtl.baseunix.WEXITSTATUS` (148)

W_EXITCODE

Synopsis: Construct an exit status based on an return code and signal.

Declaration: `function W_EXITCODE(ReturnCode: Integer; Signal: Integer) : Integer`

Visibility: default

Description: `W_EXITCODE` combines `ReturnCode` and `Signal` to a status code fit for `WaitPid`.

See also: `#rtl.baseunix.FpWaitPID` (147), `WaitProcess` (1270), `#rtl.baseunix.WTERMSIG` (149), `#rtl.baseunix.WSTOPSIG` (149), `#rtl.baseunix.WIFEXITED` (148), `WIFSTOPPED` (1271), `#rtl.baseunix.WIFSIGNALLED` (148), `W_EXITCODE` (1271), `W_STOPCODE` (1271), `#rtl.baseunix.WEXITSTATUS` (148)

W_STOPCODE

Synopsis: Construct an exit status based on a signal.

Declaration: `function W_STOPCODE(Signal: Integer) : Integer`

Visibility: default

Description: `W_STOPCODE` constructs an exit status based on `Signal`, which will cause `WIFSIGNALLED` (148) to return `True`

See also: `#rtl.baseunix.FpWaitPID` (147), `WaitProcess` (1270), `#rtl.baseunix.WTERMSIG` (149), `#rtl.baseunix.WSTOPSIG` (149), `#rtl.baseunix.WIFEXITED` (148), `WIFSTOPPED` (1271), `#rtl.baseunix.WIFSIGNALLED` (148), `W_EXITCODE` (1271), `#rtl.baseunix.WEXITSTATUS` (148)

Chapter 33

Reference for unit 'unixtype'

33.1 Overview

The `unixtype` unit contains the definitions of basic unix types. It was initially implemented by Marco van de Voort.

When porting to a new unix platform, this unit should be adapted to the sizes and conventions of the platform to which the compiler is ported.

33.2 Constants, types and variables

Constants

`ARG_MAX = 131072`

Max number of command-line arguments.

`NAME_MAX = 255`

Max length (in bytes) of filename

`PATH_MAX = 4095`

Max length (in bytes) of pathname

`Prio_PGrp = 1`

`Prio_Process = 0`

`Prio_User = 2`

`SIG_MAXSIG = 128`

Maximum signal number.

`SYS_NMLN = 65`

Max system namelength

```
_PTHREAD_MUTEX_ADAPTIVE_NP = 3
```

Mutex options:

```
_PTHREAD_MUTEX_DEFAULT = _PTHREAD_MUTEX_NORMAL
```

Mutex options:

```
_PTHREAD_MUTEX_ERRORCHECK = _PTHREAD_MUTEX_ERRORCHECK_NP
```

Mutex options:

```
_PTHREAD_MUTEX_ERRORCHECK_NP = 2
```

Mutex options: double lock returns an error code.

```
_PTHREAD_MUTEX_FAST_NP = _PTHREAD_MUTEX_ADAPTIVE_NP
```

Mutex options: Fast mutex

```
_PTHREAD_MUTEX_NORMAL = _PTHREAD_MUTEX_TIMED_NP
```

Mutex options:

```
_PTHREAD_MUTEX_RECURSIVE = _PTHREAD_MUTEX_RECURSIVE_NP
```

Mutex options:

```
_PTHREAD_MUTEX_RECURSIVE_NP = 1
```

Mutex options: recursive mutex

```
_PTHREAD_MUTEX_TIMED_NP = 0
```

Mutex options: ?

Types

```
cInt = LongInt
```

C type: integer (natural size)

```
cInt16 = SmallInt
```

C type: 16 bits sized, signed integer.

```
cInt32 = LongInt
```

C type: 32 bits sized, signed integer.

```
cInt64 = Int64
```

C type: 64 bits sized, signed integer.

cInt8 = ShortInt

C type: 8 bits sized, signed integer.

clock_t = cuLong

Clock ticks type

cLong = LongInt

C type: long signed integer (double sized)

clanglong = Int64

C type: 64-bit (double long) signed integer.

cshort = SmallInt

C type: short signed integer (half sized)

cuchar = Byte

C type: unsigned character

cUInt = Cardinal

C type: unsigned integer (natural size)

cUInt16 = Word

C type: 16 bits sized, unsigned integer.

cUInt32 = cardinal

C type: 32 bits sized, unsigned integer.

cUInt64 = qword

C type: 64 bits sized, unsigned integer.

cUInt8 = Byte

C type: 8 bits sized, unsigned integer.

cuLong = Cardinal

C type: long unsigned integer (double sized)

culonglong = qword

C type: 64-bit (double long) unsigned integer.

cushort = Word

C type: short unsigned integer (half sized)

`dev_t = cUInt64`

Device descriptor type.

`gid_t = cUInt32`

Group ID type.

`ino_t = cLong`

Inode type.

`kDev_t = cushort`

Kernel device type

`mode_t = cUInt32`

Inode mode type.

`nlink_t = cUInt32`

Number of links type.

`off_t = cInt`

Offset type.

`pcInt = ^cInt`

Pointer to `cInt` (1273) type.

`pClock = ^clock_t`

Pointer to `TClock` (1279) type.

`pcLong = ^cLong`

Pointer to `cLong` (1274) type.

`pcshort = ^cshort`

Pointer to `cShort` (1274) type.

`pcUInt = ^cUInt`

Pointer to `cUInt` (1274) type.

`pculong = ^cuLong`

Pointer to `cuLong` (1274) type.

`pcushort = ^cushort`

Pointer to cuShort (1275) type.

pDev = ^dev_t

Pointer to TDev (1279) type.

pGid = ^gid_t

Pointer to TGid (1279) type.

pid_t = cInt32

Process ID type.

pIno = ^ino_t

Pointer to TIno (1280) type.

pkDev = ^kDev_t

Pointer to TkDev (1280) type.

pMode = ^mode_t

Pointer to TMode (1280) type.

pnLink = ^nlink_t

Pointer to TnLink (1280) type.

pOff = ^off_t

Pointer to TOff (1280) type.

pPid = ^pid_t

Pointer to TPid (1280) type.

pSize = ^size_t

Pointer to TSize (1280) type.

pSockLen = ^socklen_t

Pointer to TSockLen (1280) type.

pSSize = ^ssize_t

Pointer to TsSize (1280) type

PStatFS = ^TStatfs

Pointer to TStatFS (1280) type.

```
pthread_attr_t = record
  __detachstate : cInt;
  __schedpolicy : cInt;
  __schedparam : sched_param;
  __inheritsched : cInt;
  __scope : cInt;
  __guardsize : size_t;
  __stackaddr_set : cInt;
  __stackaddr : pointer;
  __stacksize : size_t;
end
```

pthread_attr_t describes the thread attributes. It should be considered an opaque record, the names of the fields can change anytime. Use the appropriate functions to set the thread attributes.

```
pthread_condattr_t = record
  __dummy : cInt;
end
```

pthread_condattr_t describes the attributes of a thread mutex. It should be considered an opaque record, the names of the fields can change anytime.

```
pthread_cond_t = record
  __c_lock : _pthread_fastlock;
  __c_waiting : pointer;
  __padding : Array[0..48-1-sizeof(_pthread_fastlock)-sizeof(pointer)-sizeof(clonglong)];
  __align : clonglong;
end
```

pthread_cond_t describes a thread conditional variable. It should be considered an opaque record, the names of the fields can change anytime.

```
pthread_key_t = cUInt
```

Thread local storage key (opaque)

```
pthread_mutexattr_t = record
  __mutexkind : cInt;
end
```

pthread_mutexattr_t describes the attributes of a thread mutex. It should be considered an opaque record, the names of the fields can change anytime.

```
pthread_mutex_t = record
  __m_reserved : cInt;
  __m_count : cInt;
  __m_owner : pointer;
  __m_kind : cInt;
  __m_lock : _pthread_fastlock;
end
```

`_pthread_mutex_t` describes a thread mutex. It should be considered an opaque record, the names of the fields can change anytime.

```
pthread_rwlockattr_t = record
  __lockkind : cInt;
  __pshared : cInt;
end
```

`pthread_rwlockattr_t` describes the attributes of a lock. It should be considered an opaque record, the names of the fields can change anytime.

```
pthread_rwlock_t = record
  __rw_readers : cInt;
  __rw_writer : pointer;
  __rw_read_waiting : pointer;
  __rw_write_waiting : pointer;
  __rw_kind : cInt;
  __rw_pshared : cInt;
end
```

`pthread_rwlock_t` describes a lock. It should be considered an opaque record, the names of the fields can change anytime.

```
pthread_t = cuLong
```

Thread description record

```
pTime = ^time_t
```

Pointer to TTime (1280) type.

```
ptimespec = ^timespec
```

Pointer to timespec (1279) record.

```
ptimeval = ^timeval
```

Pointer to timeval (1279) record.

```
ptime_t = ^time_t
```

Pointer to time_t (1279) type.

```
pUId = ^uid_t
```

Pointer to TUid (1281) type.

```
sched_param = record
  __sched_priority : cInt;
end
```

Scheduling parameter description record.

```
sem_t = record
  __sem_lock : _pthread_fastlock;
  __sem_value : cInt;
  __sem_waiting : pointer;
end
```

`sem_t` describes a thread semaphore. It should be considered an opaque record, the names of the fields can change anytime.

```
size_t = cUInt32
```

Size specification type.

```
socklen_t = cUInt32
```

Socket address length type.

```
ssize_t = cInt32
```

Small size type.

```
TClock = clock_t
```

Alias for `clock_t` ([1274](#)) type.

```
TDev = dev_t
```

Alias for `dev_t` ([1275](#)) type.

```
TGid = gid_t
```

Alias for `gid_t` ([1275](#)) type.

```
timespec = packed record
  tv_sec : time_t;
  tv_nsec : cLong;
end
```

Record specifying time interval.

```
timeval = packed record
  tv_sec : cLong;
  tv_usec : cLong;
end
```

Time specification type.

```
time_t = cLong
```

Time span type

```
TIno = ino_t
```

Alias for `ino_t` (1275) type.

`TkDev = kDev_t`

Alias for `kDev_t` (1275) type.

`TMode = mode_t`

Alias for `mode_t` (1275) type.

`TnLink = nlink_t`

Alias for `nlink_t` (1275) type.

`TOff = off_t`

Alias for `off_t` (1275) type.

`TPid = pid_t`

Alias for `pid_t` (1276) type.

`TSize = size_t`

Alias for `size_t` (1279) type

`TSockLen = socklen_t`

Alias for `socklen_t` (1279) type.

`TSSize = ssize_t`

Alias for `ssize_t` (1279) type

```
TStatfs = packed record
  fstype : cInt;
  bsize : cInt;
  blocks : cLong;
  bfree : cLong;
  bavail : cLong;
  files : cLong;
  ffree : cLong;
  fsid : Array[0..1] of cInt;
  namelen : cLong;
  spare : Array[0..5] of cLong;
end
```

Record describing a file system in the `baseunix.fpstatfs` (1272) call.

`TTime = time_t`

Alias for `TTime` (1280) type.

`TTimeSpec = timespec`

Alias for TimeSpec (1279) type.

```
TTimeVal = timeval
```

Alias for TimeVal (1279) record.

```
TUId = uid_t
```

Alias for uid_t (1281) type.

```
uid_t = cUInt32
```

User ID type

```
_pthread_fastlock = record
  __status : cLong;
  __spinlock : cInt;
end
```

`_pthread_fastlock` describes a thread mutex. It should be considered an opaque record, the names of the fields can change anytime.

Chapter 34

Reference for unit 'unixutil'

34.1 Overview

The UnixUtil unit contains some of the routines that were present in the old Linux unit, but which do not really belong in the unix ([1234](#)) or baseunix ([71](#)) units.

Most of the functions described here have cross-platform counterparts in the SysUtils ([1071](#)) unit. It is therefore recommended to use that unit.

34.2 Constants, types and variables

Types

ComStr =

Command-line string type.

DirStr =

Filename directory part string type.

ExtStr =

Filename extension part string type.

NameStr =

Filename name part string type.

PathStr =

Filename full path string type.

Variables

Tzseconds : LongInt

Seconds west of GMT

34.3 Procedures and functions

ArrayStringToPPchar

Synopsis: Convert an array of string to an array of null-terminated strings

Declaration: `function ArrayStringToPPchar(const S: Array[] of AnsiString;
reserveentries: LongInt) : ppchar`

Visibility: default

Description: `ArrayStringToPPchar` creates an array of null-terminated strings that point to strings which are the same as the strings in the array `S`. The function returns a pointer to this array. The array and the strings it contains must be disposed of after being used, because it they are allocated on the heap.

The `ReserveEntries` parameter tells `ArrayStringToPPchar` to allocate room at the end of the array for another `ReserveEntries` entries.

Errors: If not enough memory is available, an error may occur.

See also: `StringToPPChar` ([1288](#))

Basename

Synopsis: Return basename of a file

Declaration: `function Basename(const path: PathStr;const suf: PathStr) : PathStr`

Visibility: default

Description: Returns the filename part of `Path`, stripping off `Suf` if it exists. The filename part is the whole name if `Path` contains no slash, or the part of `Path` after the last slash. The last character of the result is not a slash, unless the directory is the root directory.

Errors: None.

See also: `DirName` ([1284](#))

Listing: `./unutilx/ex48.pp`

Program `Example48`;

{ Program to demonstrate the BaseName function. }

Uses `Unix, UnixUtil`;

Var `S : String`;

begin

`S:=FExpand(Paramstr(0));`

`Writeln ('This program is called : ',Basename(S,''));`

end.

Dirname

Synopsis: Extract directory part from filename

Declaration: `function Dirname(const path: PathStr) : PathStr`

Visibility: default

Description: Returns the directory part of `Path`. The directory is the part of `Path` before the last slash, or empty if there is no slash. The last character of the result is not a slash, unless the directory is the root directory.

Errors: None.

See also: `BaseName` ([1283](#))

Listing: `./unutilib/ex47.pp`

Program `Example47;`

{ Program to demonstrate the DirName function. }

Uses `Unix, UnixUtil;`

Var `S : String;`

begin

`S:=FExpand(Paramstr(0));`

`WriteLn ('This program is in directory : ',Dirname(S));`

end.

EpochToLocal

Synopsis: Convert epoch time to local time

Declaration: `procedure EpochToLocal(epoch: LongInt; var year: Word; var month: Word;
var day: Word; var hour: Word; var minute: Word;
var second: Word)`

Visibility: default

Description: Converts the epoch time (=Number of seconds since 00:00:00 , January 1, 1970, corrected for your time zone) to local date and time.

This function takes into account the timzone settings of your system.

Errors: None

See also: `LocalToEpoch` ([1287](#))

Listing: `./unutilib/ex3.pp`

Program `Example3;`

{ Program to demonstrate the EpochToLocal function. }

Uses `Unix, UnixUtil;`

Var `Year, month, day, hour, minute, seconds : Word;`

```

begin
  EpochToLocal ( GetEpochTime , Year , month , day , hour , minute , seconds );
  Writeln ( ' Current date : ', Day:2 , '/' , Month:2 , '/' , Year:4 );
  Writeln ( ' Current time : ', Hour:2 , ':' , minute:2 , ':' , seconds:2 );
end.

```

FNMatch

Synopsis: Check whether filename matches wildcard specification

Declaration: `function FNMatch(const Pattern: String;const Name: String) : Boolean`

Visibility: default

Description: `FNMatch` returns `True` if the filename in `Name` matches the wildcard pattern in `Pattern`, `False` otherwise.

`Pattern` can contain the wildcards `*` (match zero or more arbitrary characters) or `?` (match a single character).

Errors: None.

See also: `#rtl.unix.FSearch` ([1261](#))

Listing: `./unutilx/ex69.pp`

Program Example69;

{ Program to demonstrate the FNMatch function. }

Uses unixutil;

Procedure TestMatch(Pattern ,Name : String);

```

begin
  Write ( ' " ', Name, ' " ');
  If FNMatch ( Pattern ,Name) then
    Write ( ' matches ')
  else
    Write ( ' does not match ');
  Writeln( ' " ', Pattern, ' " ');
end;

begin
  TestMatch( '* ', 'FileName ');
  TestMatch( '. * ', 'FileName ');
  TestMatch( '* a * ', 'FileName ');
  TestMatch( '? ile * ', 'FileName ');
  TestMatch( '? ', 'FileName ');
  TestMatch( '. ? ', 'FileName ');
  TestMatch( '? a * ', 'FileName ');
  TestMatch( ' ?? * me? ', 'FileName ');
end.

```

FSplit

Synopsis: Split filename into path, name and extension

Declaration: `procedure FSplit(const Path: PathStr; var Dir: DirStr; var Name: NameStr;
var Ext: ExtStr)`

Visibility: default

Description: `FSplit` splits a full file name into 3 parts : A Path, a Name and an extension (in `ext`). The extension is taken to be all letters after the last dot (.).

Errors: None.

See also: `#rtl.unix.FSearch` ([1261](#))

Listing: `./unutilx/ex67.pp`

Program `Example67;`

`uses UnixUtil;`

`{ Program to demonstrate the FSplit function. }`

`var`

`Path, Name, Ext : string;`

`begin`

`FSplit(ParamStr(1), Path, Name, Ext);`

`WriteLn('Split ', ParamStr(1), ' in:');`

`WriteLn('Path : ', Path);`

`WriteLn('Name : ', Name);`

`WriteLn('Extension : ', Ext);`

`end.`

GetFS

Synopsis: Return file selector

Declaration: `function GetFS(var T: Text) : LongInt`
`function GetFS(var F: File) : LongInt`

Visibility: default

Description: `GetFS` returns the file selector that the kernel provided for your file. In principle you don't need this file selector. Only for some calls it is needed, such as the `#rtl.baseunix.fpSelect` ([131](#)) call or so.

Errors: In case the file was not opened, then -1 is returned.

See also: `#rtl.baseunix.fpSelect` ([131](#))

Listing: `./unutilx/ex34.pp`

Program `Example33;`

`{ Program to demonstrate the SelectText function. }`

`Uses Unix;`

```

Var tv : TimeVal;

begin
  Writeln ( 'Press the <ENTER> to continue the program.' );
  { Wait until File descriptor 0 (=Input) changes }
  SelectText ( Input, nil );
  { Get rid of <ENTER> in buffer }
  readln;
  Writeln ( 'Press <ENTER> key in less than 2 seconds...' );
  tv.tv_sec:=2;
  tv.tv_sec:=0;
  if SelectText ( Input, @tv)>0 then
    Writeln ( 'Thank you !' )
  else
    Writeln ( 'Too late !' );
end.

```

GregorianToJulian

Synopsis: Converts a gregorian date to a julian date

Declaration: `function GregorianToJulian(Year: LongInt;Month: LongInt;Day: LongInt)
: LongInt`

Visibility: default

Description: `GregorianToJulian` takes a gregorian date and converts it to a Julian day.

Errors: None.

See also: `JulianToGregorian` ([1287](#))

JulianToGregorian

Synopsis: Converts a julian date to a gregorian date

Declaration: `procedure JulianToGregorian(JulianDN: LongInt;var Year: Word;
var Month: Word;var Day: Word)`

Visibility: default

Description: `JulianToGregorian` takes a julian day and converts it to a gregorian date. (Start of the Julian Date count is from 0 at 12 noon 1 JAN -4712 (4713 BC),)

Errors: None.

See also: `GregorianToJulian` ([1287](#))

LocalToEpoch

Synopsis: Convert local time to epoch (unix) time

Declaration: `function LocalToEpoch(year: Word;month: Word;day: Word;hour: Word;
minute: Word;second: Word) : LongInt`

Visibility: default

Description: Converts the Local time to epoch time (=Number of seconds since 00:00:00 , January 1, 1970).

Errors: None

See also: EpochToLocal ([1284](#))

Listing: ./unutilex/ex4.pp

Program Example4:

```
{ Program to demonstrate the LocalToEpoch function. }
```

Uses UnixUtil;

```
Var year , month , day , hour , minute , second : Word;
```

begin

```
Write ( 'Year      : ' ); readln (Year);
```

```
Write ( 'Month' : ' '); readln (Month);
```

```
Write ( 'Day' : ' '); readln (Day);
```

```
Write ( 'Hour' : ' '); readIn (Hour);
```

```
Write ( 'Minute' : ' '); readIn ( Minute );
```

```
Write ( 'Seonds : ' ); readln ( Second );
```

```
Write ( 'This is : ' );
```

```
Write (LocalToEpoch(year, month, day, hour, minute, second));
```

```
WriteIn ( ' seconds past 00:00 1/1/1980' );
```

end.

StringToPPChar

Synopsis: Split string in list of null-terminated strings

```
Declaration: function StringToPPChar(S: PChar; ReserveEntries: Integer) : ppchar
            function StringToPPChar(var S: String; ReserveEntries: Integer) : ppchar
            function StringToPPChar(var S: AnsiString; ReserveEntries: Integer)
                : ppchar
```

Visibility: default

Description: `StringToPPChar` splits the string `S` in words, replacing any whitespace with zero characters. It returns a pointer to an array of `pchars` that point to the first letters of the words in `S`. This array is terminated by a `Nil` pointer.

The function does *not* add a zero character to the end of the string unless it ends on whitespace.

The function reserves memory on the heap to store the array of `PChar`; The caller is responsible for freeing this memory.

This function can be called to create arguments for the various Exec calls.

Errors: None.

See also: [ArrayStringToPPchar \(1283\)](#), [#rtl.baseunix.FpExecve \(106\)](#)

Listing: ./unutilx/ex70.pp

Program Example70;

```
{ Program to demonstrate the StringToPPchar function. }
```

Uses UnixUtil;

Var S : **String**;
 P : PPChar;
 I : longint;

begin
 // remark whitespace at end.
 S:= 'This is a string with words. ';
 P:=StringToPPChar(S,0);
 I:=0;
 While P[i]<>Nil **do**
 begin
 Writeln('Word ',i, ' : ',P[i]);
 Inc(I);
 end;
 FreeMem(P, i***SizeOf**(Pchar));
end.

Chapter 35

Reference for unit 'Video'

35.1 Examples utility unit

The examples in this section make use of the unit `vidutil`, which contains the `TextOut` function. This function writes a text to the screen at a given location. It looks as follows:

35.2 Writing a custom video driver

Writing a custom video driver is not difficult, and generally means implementing a couple of functions, which should be registered with the `SetVideoDriver` ([1306](#)) function. The various functions that can be implemented are located in the `TVideoDriver` ([1296](#)) record:

```
TVideoDriver = Record
  InitDriver      : Procedure;
  DoneDriver      : Procedure;
  UpdateScreen    : Procedure(Force : Boolean);
  ClearScreen     : Procedure;
  SetVideoMode    : Function (Const Mode : TVideoMode) : Boolean;
  GetVideoModeCount : Function : Word;
  GetVideoModeData : Function(Index : Word; Var Data : TVideoMode) : Boolean;
  SetCursorPos    : procedure (NewCursorX, NewCursorY: Word);
  GetCursorType   : function : Word;
  SetCursorType   : procedure (NewType: Word);
  GetCapabilities : Function : Word;
end;
```

Not all of these functions must be implemented. In fact, the only absolutely necessary function to write a functioning driver is the `UpdateScreen` function. The general calls in the `Video` unit will check which functionality is implemented by the driver.

The functionality of these calls is the same as the functionality of the calls in the `video` unit, so the expected behaviour can be found in the previous section. Some of the calls, however, need some additional remarks.

InitDriver Called by `InitVideo`, this function should initialize any data structures needed for the functionality of the driver, maybe do some screen initializations. The function is guaranteed to be called only once; It can only be called again after a call to `DoneVideo`. The variables

`ScreenWidth` and `ScreenHeight` should be initialized correctly after a call to this function, as the `InitVideo` call will initialize the `VideoBuf` and `OldVideoBuf` arrays based on their values.

DoneDriver This should clean up any structures that have been initialized in the `InitDriver` function. It should possibly also restore the screen as it was before the driver was initialized. The `VideoBuf` and `OldVideoBuf` arrays will be disposed of by the general `DoneVideo` call.

UpdateScreen This is the only required function of the driver. It should update the screen based on the `VideoBuf` array's contents. It can optimize this process by comparing the values with values in the `OldVideoBuf` array. After updating the screen, the `UpdateScreen` procedure should update the `OldVideoBuf` by itself. If the `Force` parameter is `True`, the whole screen should be updated, not just the changed values.

ClearScreen If there is a faster way to clear the screen than to write spaces in all character cells, then it can be implemented here. If the driver does not implement this function, then the general routines will write spaces in all video cells, and will call `UpdateScreen(True)`.

SetVideoMode Should set the desired video mode, if available. It should return `True` if the mode was set, `False` if not.

GetVideoModeCount Should return the number of supported video modes. If no modes are supported, this function should not be implemented; the general routines will return 1. (for the current mode)

GetVideoModeData Should return the data for the `Index`-th mode; `Index` is zero based. The function should return `true` if the data was returned correctly, `false` if `Index` contains an invalid index. If this is not implemented, then the general routine will return the current video mode when `Index` equals 0.

GetCapabilities If this function is not implemented, zero (i.e. no capabilities) will be returned by the general function.

The following unit shows how to override a video driver, with a driver that writes debug information to a file. The unit can be used in any of the demonstration programs, by simply including it in the `uses` clause. Setting `DetailedVideoLogging` to `True` will create a more detailed log (but will also slow down functioning)

35.3 Overview

The `Video` unit implements a screen access layer which is system independent. It can be used to write on the screen in a system-independent way, which should be optimal on all platforms for which the unit is implemented.

The working of the `Video` is simple: After calling `InitVideo` (1304), the array `VideoBuf` contains a representation of the video screen of size `ScreenWidth*ScreenHeight`, going from left to right and top to bottom when walking the array elements: `VideoBuf[0]` contains the character and color code of the top-left character on the screen. `VideoBuf[ScreenWidth]` contains the data for the character in the first column of the second row on the screen, and so on.

To write to the 'screen', the text to be written should be written to the `VideoBuf` array. Calling `UpdateScreen` (1307) will then cp the text to the screen in the most optimal way. (an example can be found further on).

The color attribute is a combination of the foreground and background color, plus the blink bit. The bits describe the various color combinations:

bits 0-3 The foreground color. Can be set using all color constants.

bits 4-6 The background color. Can be set using a subset of the color constants.

bit 7 The blinking bit. If this bit is set, the character will appear blinking.

Each possible color has a constant associated with it, see the constants section for a list of constants.

The foreground and background color can be combined to a color attribute with the following code:

```
Attr:=ForeGroundColor + (BackGroundColor shl 4);
```

The color attribute can be logically or-ed with the blink attribute to produce a blinking character:

```
Attr:=Attr or blink;
```

But not all drivers may support this.

The contents of the `VideoBuf` array may be modified: This is 'writing' to the screen. As soon as everything that needs to be written in the array is in the `VideoBuf` array, calling `UpdateScreen` will copy the contents of the array screen to the screen, in a manner that is as efficient as possible.

The updating of the screen can be prohibited to optimize performance; To this end, the `LockScreenUpdate` (1304) function can be used: This will increment an internal counter. As long as the counter differs from zero, calling `UpdateScreen` (1307) will not do anything. The counter can be lowered with `UnlockScreenUpdate` (1307). When it reaches zero, the next call to `UpdateScreen` (1307) will actually update the screen. This is useful when having nested procedures that do a lot of screen writing.

The video unit also presents an interface for custom screen drivers, thus it is possible to override the default screen driver with a custom screen driver, see the `SetVideoDriver` (1306) call. The current video driver can be retrieved using the `GetVideoDriver` (1301) call.

Remark: The video unit should *not* be used together with the CRT unit. Doing so will result in very strange behaviour, possibly program crashes.

35.4 Constants, types and variables

Constants

`Black = 0`

Black color attribute

`Blink = 128`

Blink attribute

`Blue = 1`

Blue color attribute

`Brown = 6`

Brown color attribute

`cpBlink = $0002`

Video driver supports blink attribute

`cpChangeCursor = $0020`

Video driver supports changing cursor shape.

`cpChangeFont = $0008`

Video driver supports changing screen font.

`cpChangeMode = $0010`

Video driver supports changing mode

`cpColor = $0004`

Video driver supports color

`cpUnderLine = $0001`

Video driver supports underline attribute

`crBlock = 2`

Block cursor

`crHalfBlock = 3`

Half block cursor

`crHidden = 0`

Hide cursor

`crUnderLine = 1`

Underline cursor

`Cyan = 3`

Cyan color attribute

`DarkGray = 8`

Dark gray color attribute

`errOk = 0`

No error

`ErrorCode : LongInt = ErrOK`

Error code returned by the last operation.

`ErrorHandler : TErrorHandler = @DefaultErrorHandler`

The `ErrorHandler` variable can be set to a custom-error handling function. It is set by default to the `DefaultErrorHandler` (1298) function.

`ErrorInfo : Pointer = nil`

Pointer to extended error information.

`errVioBase = 1000`

Base value for video errors

`errVioInit = errVioBase + 1`

Video driver initialization error.

`errVioNoSuchMode = errVioBase + 3`

Invalid video mode

`errVioNotSupported = errVioBase + 2`

Unsupported video function

`FVMaxWidth = 132`

Maximum screen buffer width.

`Green = 2`

Green color attribute

`LightBlue = 9`

Light Blue color attribute

`LightCyan = 11`

Light cyan color attribute

`LightGray = 7`

Light gray color attribute

`LightGreen = 10`

Light green color attribute

`LightMagenta = 13`

Light magenta color attribute

`LightRed = 12`

Light red color attribute

LowAscii : Boolean = true

Only use low ascii characters

Magenta = 5

Magenta color attribute

NoExtendedFrame : Boolean = false

Disable transformation of control characters on unix terminals

Red = 4

Red color attribute

ScreenHeight : Word = 0

Current screen height

ScreenWidth : Word = 0

Current screen Width

vioOK = 0

No errors occurred

White = 15

White color attribute

Yellow = 14

Yellow color attribute

Types

PVideoBuf = ^TVideoBuf

Pointer type to TVideoBuf ([1296](#))

PVideoCell = ^TVideoCell

Pointer type to TVideoCell ([1296](#))

PVideoMode = ^TVideoMode

Pointer to TVideoMode ([1297](#)) record.

```
TErrorHandler = function(Code: LongInt; Info: Pointer)
                    : TErrorHandlerReturnValue
```

Table 35.1: Enumeration values for type TErrorHandlerReturnValue

Value	Explanation
errAbort	abort and return error code
errContinue	abort without returning an errorcode.
errRetry	retry the operation

The TErrorHandler function is used to register an own error handling function. It should be used when installing a custom error handling function, and must return one of the above values.

Code should contain the error code for the error condition, and the Info parameter may contain any data type specific to the error code passed to the function.

```
TErrorHandlerReturnValue = (errRetry, errAbort, errContinue)
```

Type used to report and respond to error conditions

```
TVideoBuf = Array[0..32759] of TVideoCell
```

The TVideoBuf type represents the screen.

```
TVideoCell = Word
```

TVideoCell describes one character on the screen. One of the bytes contains the color attribute with which the character is drawn on the screen, and the other byte contains the ASCII code of the character to be drawn. The exact position of the different bytes in the record is operating system specific. On most little-endian systems, the high byte represents the color attribute, while the low-byte represents the ASCII code of the character to be drawn.

```
TVideoDriver = record
  InitDriver : procedure;
  DoneDriver : procedure;
  UpdateScreen : procedure(Force: Boolean);
  ClearScreen : procedure;
  SetVideoMode : function(const Mode: TVideoMode) : Boolean;
  GetVideoModeCount : function : Word;
  GetVideoModeData : function(Index: Word; var Data: TVideoMode) : Boolean;
  SetCursorPos : procedure(NewCursorX: Word; NewCursorY: Word);
  GetCursorType : function : Word;
  SetCursorType : procedure(NewType: Word);
  GetCapabilities : function : Word;
end
```

TVideoDriver record can be used to install a custom video driver, with the SetVideoDriver ([1306](#)) call.

An explanation of all fields can be found there.

```
TVideoMode = record
  Col : Word;
  Row : Word;
  Color : Boolean;
end
```

The `TVideoMode` record describes a videomode. Its fields are self-explaining: `Col`, `Row` describe the number of columns and rows on the screen for this mode. `Color` is `True` if this mode supports colors, or `False` if not.

```
TVideoModeSelector = function(const VideoMode: TVideoMode;
                               Params: LongInt) : Boolean
```

Video mode selection callback prototype.

Variables

`CursorLines` : Byte

`CursorX` : Word

Current horizontal position in the screen where items will be written.

`CursorY` : Word

Current vertical position in the screen where items will be written.

`OldVideoBuf` : `PVideoBuf`

The `OldVideoBuf` contains the state of the video screen after the last screen update. The `UpdateScreen` (1307) function uses this array to decide which characters on screen should be updated, and which not.

Note that the `OldVideoBuf` array may be ignored by some drivers, so it should not be used. The Array is in the interface section of the video unit mainly so drivers that need it can make use of it.

`ScreenColor` : Boolean

`ScreenColor` indicates whether the current screen supports colors.

`VideoBuf` : `PVideoBuf`

`VideoBuf` forms the heart of the `Video` unit: This variable represents the physical screen. Writing to this array and calling `UpdateScreen` (1307) will write the actual characters to the screen.

`VideoBufSize` : LongInt

Current size of the video buffer pointed to by `VideoBuf` (1297)

35.5 Procedures and functions

ClearScreen

Synopsis: Clear the video screen.

Declaration: procedure `ClearScreen`

Visibility: default

Description: `ClearScreen` clears the entire screen, and calls `UpdateScreen` (1307) after that. This is done by writing spaces to all character cells of the video buffer in the default color (lightgray on black, color attribute `\$07`).

Errors: None.

See also: [InitVideo \(1304\)](#), [UpdateScreen \(1307\)](#)

Listing: ./videoex/ex3.pp

```
program testvideo;
```

```
uses video , keyboard , vidutil ;
```

```
{\ifnundefined cpu86}
{\error This example only works on intel 80x86 machines}
{\endif}
```

Var

```
i : longint;  
k : TkeyEvent;
```

begin

```
InitVideo;  
InitKeyboard;  
For i:=1 to 10 do  
    TextOut(i,i, 'Press any key to clear screen');  
UpdateScreen(false);  
K:=GetKeyEvent;  
ClearScreen;  
TextOut(1,1, 'Cleared screen. Press any key to end');  
UpdateScreen(true);  
K:=GetKeyEvent;  
DoneKeyBoard;  
DoneVideo;
```

end .

DefaultErrorHandler

Synopsis: Default error handling routine.

[illegible]

Visibility: default

Description: `DefaultErrorHandler` is the default error handler used by the video driver. It simply sets the error code `AErrorCode` and `AErrorInfo` in the global variables `ErrorCode` and `ErrorInfo` and returns `errContinue`.

Errors: None.

DoneVideo

Synopsis: Disable video driver.

Declaration: procedure DoneVideo

Visibility: default

Description: DoneVideo disables the Video driver if the video driver is active. If the videodriver was already disabled or not yet initialized, it does nothing. Disabling the driver means it will clean up any allocated resources, possibly restore the screen in the state it was before InitVideo was called. Particularly, the VideoBuf and OldVideoBuf arrays are no longer valid after a call to DoneVideo. The DoneVideo should always be called if InitVideo was called. Failing to do so may leave the screen in an unusable state after the program exits.

For an example, see most other functions.

Errors: Normally none. If the driver reports an error, this is done through the ErrorCode variable.

See also: InitVideo ([1304](#))

GetCapabilities

Synopsis: Get current driver capabilities.

Declaration: function GetCapabilities : Word

Visibility: default

Description: GetCapabilities returns the capabilities of the current driver. It is an or-ed combination of the following constants:

cpUnderLineVideo driver supports underline attribute

cpBlinkVideo driver supports blink attribute

cpColorVideo driver supports color

cpChangeFontVideo driver supports changing screen font.

cpChangeModeVideo driver supports changing mode

cpChangeCursorVideo driver supports changing cursor shape.

Note that the video driver should not yet be initialized to use this function. It is a property of the driver.

Errors: None.

See also: GetCursorType ([1300](#)), GetVideoDriver ([1301](#))

Listing: ./videoex/ex4.pp

Program Example4;

{ Program to demonstrate the GetCapabilities function. }

Uses video;

Var

W: Word;

Procedure TestCap(Cap: Word; Msg : **String**);

begin

Write(Msg, ' : ');

If (W and Cap=Cap) **then**

WriteLn('Yes')


```

    else
        Writeln( 'No' );
    end;

begin
    W:= GetCapabilities;
    Writeln( 'Video driver supports following functionality' );
    TestCap(cpUnderLine, 'Underlined characters');
    TestCap(cpBlink, 'Blinking characters');
    TestCap(cpColor, 'Color characters');
    TestCap(cpChangeFont, 'Changing font');
    TestCap(cpChangeMode, 'Changing video mode');
    TestCap(cpChangeCursor, 'Changing cursor shape');
end.

```

GetCursorType

Synopsis: Get screen cursor type

Declaration: function GetCursorType : Word

Visibility: default

Description: GetCursorType returns the current cursor type. It is one of the following values:

crHidden Hide cursor
crUnderLine Underline cursor
crBlock Block cursor
crHalfBlock Half block cursor

Note that not all drivers support all types of cursors.

Errors: None.

See also: SetCursorType ([1306](#)), GetCapabilities ([1299](#))

Listing: ./videoex/ex5.pp

Program Example5;

{ Program to demonstrate the GetCursorType function. }

Uses video, keyboard, vidutil;

Const

CursorTypes : **Array**[crHidden..crHalfBlock] **of string** =
 ('Hidden', 'UnderLine', 'Block', 'HalfBlock');

begin

```

    InitVideo;
    InitKeyboard;
    TextOut(1,1, 'Cursor type: '+CursorTypes[GetCursorType]);
    TextOut(1,2, 'Press any key to exit. ');
    UpdateScreen(False);
    GetKeyEvent;
    DoneKeyboard;
    DoneVideo;
end.

```

GetLockScreenCount

Synopsis: Get the screen lock update count.

Declaration: `function GetLockScreenCount : Integer`

Visibility: default

Description: `GetLockScreenCount` returns the current lock level. When the lock level is zero, a call to `UpdateScreen` (1307) will actually update the screen.

Errors: None.

See also: `LockScreenUpdate` (1304), `UnlockScreenUpdate` (1307), `UpdateScreen` (1307)

Listing: `./videoex/ex6.pp`

Program Example6;

{ Program to demonstrate the GetLockScreenCount function. }

Uses video, keyboard, vidutil;

Var

 I : Longint;
 S : **String**;

begin

```
  InitVideo;
  InitKeyboard;
  TextOut(1,1,'Press key till new text appears. ');
  UpdateScreen(False);
  Randomize;
  For I:=0 to Random(10)+1 do
    LockScreenUpdate;
  I:=0;
  While GetLockScreenCount<>0 do
    begin
      Inc(I);
      Str(I,S);
      UnlockScreenUpdate;
      GetKeyEvent;
      TextOut(1,1,'UnLockScreenUpdate had to be called '+S+' times ');
      UpdateScreen(False);
    end;
  TextOut(1,2,'Press any key to end. ');
  UpdateScreen(False);
  GetKeyEvent;
  DoneKeyboard;
  DoneVideo;
end.
```

GetVideoDriver

Synopsis: Get a copy of the current video driver.

Declaration: `procedure GetVideoDriver(var Driver: TVideoDriver)`

Visibility: default

Description: `GetVideoMode` returns the settings of the currently active video mode. The `row`, `col` fields indicate the dimensions of the current video mode, and `Color` is true if the current video supports colors.

Errors: None.

See also: `SetVideoMode` ([1307](#)), `GetVideoModeData` ([1304](#))

Listing: `./videoex/ex7.pp`

Program `Example7`;

{ Program to demonstrate the GetVideoMode function. }

Uses `video`, `keyboard`, `vidutil`;

Var

`M` : `TVideoMode`;

`S` : **String**;

begin

`InitVideo`;

`InitKeyboard`;

`GetVideoMode`(`M`);

if `M.Color` **then**

`TextOut`(1,1,'Current mode has color')

else

`TextOut`(1,1,'Current mode does not have color');

Str(`M.Row`,`S`);

`TextOut`(1,2,'Number of rows : '+`S`);

Str(`M.Col`,`S`);

`TextOut`(1,3,'Number of columns : '+`S`);

`Textout`(1,4,'Press any key to exit.');

`UpdateScreen`(`False`);

`GetKeyEvent`;

`DoneKeyboard`;

`DoneVideo`;

end.

GetVideoMode

Synopsis: Return current video mode

Declaration: `procedure GetVideoMode(var Mode: TVideoMode)`

Visibility: default

Description: Return current video mode

GetVideoModeCount

Synopsis: Get the number of video modes supported by the driver.

Declaration: `function GetVideoModeCount : Word`

Visibility: default

Description: `GetVideoModeCount` returns the number of video modes that the current driver supports. If the driver does not support switching of modes, then 1 is returned.

This function can be used in conjunction with the `GetVideoModeData` (1304) function to retrieve data for the supported video modes.

Errors: None.

See also: `GetVideoModeData` (1304), `GetVideoMode` (1302)

Listing: `./videoex/ex8.pp`

Program `Example8`;

{ Program to demonstrate the GetVideoModeCount function. }

Uses `video, keyboard, vidutil`;

Procedure `DumpMode (M : TVideoMode; Index : Integer)`;

Var

`S : String`;

begin

```
  Str(Index:2,S);
  inc(Index);
  TextOut(1,Index,'Data for mode '+S+' : ');
  if M.Color then
    TextOut(19,Index,'  color, ')
  else
    TextOut(19,Index,'No color, ');
  Str(M.Row:3,S);
  TextOut(28,Index,S+' rows ');
  Str(M.Col:3,S);
  TextOut(36,Index,S+' columns ');
```

end;

Var

`i, Count : Integer`;
`m : TVideoMode`;

begin

```
  InitVideo;
  InitKeyboard;
  Count:=GetVideoModeCount;
  For I:=1 to Count do
    begin
      GetVideoModeData(I-1,M);
      DumpMode(M, I-1);
    end;
  TextOut(1,Count+1,'Press any key to exit ');
  UpdateScreen(False);
  GetKeyEvent;
  DoneKeyboard;
  DoneVideo;
```

end.

GetVideoModeData

Synopsis: Get the specifications for a video mode

Declaration: `function GetVideoModeData(Index: Word; var Data: TVideoMode) : Boolean`

Visibility: default

Description: `GetVideoModeData` returns the characteristics of the `Index`-th video mode in `Data`. `Index` is zero based, and has a maximum value of `GetVideoModeCount-1`. If the current driver does not support setting of modes (`GetVideoModeCount=1`) and `Index` is zero, the current mode is returned.

The function returns `True` if the mode data was retrieved successfully, `False` otherwise.

For an example, see `GetVideoModeCount` ([1302](#)).

Errors: In case `Index` has a wrong value, `False` is returned.

See also: `GetVideoModeCount` ([1302](#)), `SetVideoMode` ([1307](#)), `GetVideoMode` ([1302](#))

InitVideo

Synopsis: Initialize video driver.

Declaration: `procedure InitVideo`

Visibility: default

Description: `InitVideo` initializes the video subsystem. If the video system was already initialized, it does nothing. After the driver has been initialized, the `VideoBuf` and `OldVideoBuf` pointers are initialized, based on the `ScreenWidth` and `ScreenHeight` variables. When this is done, the screen is cleared.

For an example, see most other functions.

Errors: if the driver fails to initialize, the `ErrorCode` variable is set.

See also: `DoneVideo` ([1298](#))

LockScreenUpdate

Synopsis: Prevent further screen updates.

Declaration: `procedure LockScreenUpdate`

Visibility: default

Description: `LockScreenUpdate` increments the screen update lock count with one. As long as the screen update lock count is not zero, `UpdateScreen` ([1307](#)) will not actually update the screen.

This function can be used to optimize screen updating: If a lot of writing on the screen needs to be done (by possibly unknown functions), calling `LockScreenUpdate` before the drawing, and `UnlockScreenUpdate` ([1307](#)) after the drawing, followed by a `UpdateScreen` ([1307](#)) call, all writing will be shown on screen at once.

For an example, see `GetLockScreenCount` ([1301](#)).

Errors: None.

See also: `UpdateScreen` ([1307](#)), `UnlockScreenUpdate` ([1307](#)), `GetLockScreenCount` ([1301](#))

SetCursorPos

Synopsis: Set write cursor position.

Declaration: `procedure SetCursorPos(NewCursorX: Word;NewCursorY: Word)`

Visibility: default

Description: `SetCursorPos` positions the cursor on the given position: Column `NewCursorX` and row `NewCursorY`. The origin of the screen is the upper left corner, and has coordinates (0 , 0).

The current position is stored in the `CursorX` and `CursorY` variables.

Errors: None.

See also: `SetCursorType` ([1306](#))

Listing: `./videoex/ex2.pp`

```

program example2;

uses video , keyboard;

{ $ifndef cpu86 }
{ $error This example only works on intel 80x86 machines }
{ $endif }

Var
  P,PP,D : Integer;
  K: TKeyEvent;

  Procedure PutSquare ( P : INteger; C : Char);

begin
  VideoBuf^[P]:=Ord(C)+($07 shl 8);
  VideoBuf^[P+ScreenWidth]:=Ord(c)+($07 shl 8);
  VideoBuf^[P+1]:=Ord(c)+($07 shl 8);
  VideoBuf^[P+ScreenWidth+1]:=Ord(c)+($07 shl 8);
end;

begin
  InitVideo;
  InitKeyBoard;
  P:=0;
  PP:=-1;
  Repeat
    If PP<>-1 then
      PutSquare(PP, ' ');
      PutSquare(P, '#');
      SetCursorPos(P Mod ScreenWidth,P div ScreenWidth);
      UpdateScreen(False);
      PP:=P;
    Repeat
      D:=0;
      K:=TranslateKeyEvent(GetKeyEvent);
      Case GetKeyEventCode(K) of
        kbdLeft : If (P Mod ScreenWidth)<>0 then
          D:=-1;
        kbdUp : If P>=ScreenWidth then
          D:=-ScreenWidth;

```

```

    kbdRight : if ((P+2) Mod ScreenWidth)<>0 then
        D:=1;
    kbdDown : if (P<(VideoBufSize div 2)-(ScreenWidth*2)) then
        D:=ScreenWidth;
    end;
    Until (D<>0) or (GetKeyEventChar(K)= 'q ');
    P:=P+D;
    until GetKeyEventChar(K)= 'q ';
    DoneKeyBoard;
    DoneVideo;
end.

```

SetCursorType

Synopsis: Set cursor type

Declaration: `procedure SetCursorType(NewType: Word)`

Visibility: default

Description: `SetCursorType` sets the cursor to the type specified in `NewType`.

crHiddenHide cursor

crUnderLineUnderline cursor

crBlockBlock cursor

crHalfBlockHalf block cursor

Errors: None.

See also: `SetCursorPos` ([1305](#))

SetVideoDriver

Synopsis: Install a new video driver.

Declaration: `function SetVideoDriver(const Driver: TVideoDriver) : Boolean`

Visibility: default

Description: `SetVideoDriver` sets the videodriver to be used to `Driver`. If the current videodriver is initialized (after a call to `InitVideo`) then it does nothing and returns `False`.

A new driver can only be installed if the previous driver was not yet activated (i.e. before a call to `InitVideo` ([1304](#))) or after it was deactivated (i.e after a call to `DoneVideo`).

For more information about installing a videodriver, see `viddriver` ([1290](#)).

For an example, see the section on writing a custom video driver.

Errors: If the current driver is initialized, then `False` is returned.

See also: `viddriver` ([1290](#))

SetVideoMode

Synopsis: Set current video mode.

Declaration: `function SetVideoMode(const Mode: TVideoMode) : Boolean`

Visibility: default

Description: `SetVideoMode` sets the video mode to the mode specified in `Mode`:

If the call was succesful, then the screen will have `Col` columns and `Row` rows, and will be displaying in color if `Color` is `True`.

The function returns `True` if the mode was set succesfully, `False` otherwise.

Note that the video mode may not always be set. E.g. a console on Linux or a telnet session cannot always set the mode. It is important to check the error value returned by this function if it was not succesful.

The mode can be set when the video driver has not yet been initialized (i.e. before `InitVideo` (1304) was called) In that case, the video mode will be stored, and after the driver was initialized, an attempt will be made to set the requested mode. Changing the video driver before the call to `InitVideo` will clear the stored video mode.

To know which modes are valid, use the `GetVideoModeCount` (1302) and `GetVideoModeData` (1304) functions. To retrieve the current video mode, use the `GetVideoMode` (1302) procedure.

Errors: If the specified mode cannot be set, then `errVioNoSuchMode` may be set in `ErrorCode`

See also: `GetVideoModeCount` (1302), `GetVideoModeData` (1304), `GetVideoMode` (1302)

UnlockScreenUpdate

Synopsis: Unlock screen update.

Declaration: `procedure UnlockScreenUpdate`

Visibility: default

Description: `UnlockScreenUpdate` decrements the screen update lock count with one if it is larger than zero.

When the lock count reaches zero, the `UpdateScreen` (1307) will actually update the screen. No screen update will be performed as long as the screen update lock count is nonzero. This mechanism can be used to increase screen performance in case a lot of writing is done.

It is important to make sure that each call to `LockScreenUpdate` (1304) is matched by exactly one call to `UnlockScreenUpdate`

For an example, see `GetLockScreenCount` (1301).

Errors: None.

See also: `LockScreenUpdate` (1304), `GetLockScreenCount` (1301), `UpdateScreen` (1307)

UpdateScreen

Synopsis: Update physical screen with internal screen image.

Declaration: `procedure UpdateScreen(Force: Boolean)`

Visibility: default

Description: `UpdateScreen` synchronizes the actual screen with the contents of the `VideoBuf` internal buffer. The parameter `Force` specifies whether the whole screen has to be redrawn (`Force=True`) or only parts that have changed since the last update of the screen.

The `Video` unit keeps an internal copy of the screen as it last wrote it to the screen (in the `OldVideoBuf` array). The current contents of `VideoBuf` are examined to see what locations on the screen need to be updated. On slow terminals (e.g. a linux telnet session) this mechanism can speed up the screen redraw considerably.

For an example, see most other functions.

Errors: None.

See also: `ClearScreen` ([1297](#))

Chapter 36

Reference for unit 'x86'

36.1 Used units

Table 36.1: Used units by unit 'x86'

Name	Page
BaseUnix	71

36.2 Overview

The x86 unit contains some of the routines that were present in the 1.0.X Linux unit, and which were Intel (PC) architecture specific.

These calls have been preserved for compatibility, but should be considered deprecated: they are not portable and may not even work on future linux versions.

36.3 Procedures and functions

fplOperm

Synopsis: Set permission on IO ports

Declaration: `function fplOperm(From: Cardinal; Num: Cardinal; Value: cInt) : cInt`

Visibility: default

Description: `fplOperm` sets permissions on `Num` ports starting with port `From` to `Value`. The function returns zero if the call was successful, a nonzero value otherwise.

Note:

- This works ONLY as root.
- Only the first 0x03ff ports can be set.
- When doing a `FpFork` ([109](#)), the permissions are reset. When doing a `FpExecVE` ([106](#)) they are kept.

Errors: Extended error information can be retrieved with `FpGetErrno` ([112](#))

fpIoPL

Synopsis: Set I/O privilege level

Declaration: `function fpIoPL(Level: cInt) : cInt`

Visibility: default

Description: `FpIoPL` sets the I/O privilege level. It is intended for completeness only, one should normally not use it.

ReadPort

Synopsis: Read data from a PC port

Declaration: `procedure ReadPort(Port: LongInt; var Value: Byte)`
`procedure ReadPort(Port: LongInt; var Value: LongInt)`
`procedure ReadPort(Port: LongInt; var Value: Word)`

Visibility: default

Description: `ReadPort` reads one Byte, Word or Longint from port `Port` into `Value`.

Note that you need permission to read a port. This permission can be set by the root user with the `FpIOPerm` (1309) call.

Errors: In case of an error (not enough permissions read this port), runtime 216 (*Access Violation*) will occur.

See also: `FpIOPerm` (1309), `ReadPortB` (1310), `ReadPortW` (1311), `ReadPortL` (1311), `WritePort` (1311), `WritePortB` (1312), `WritePortL` (1312), `WritePortW` (1312)

ReadPortB

Synopsis: Read bytes from a PC port

Declaration: `function ReadPortB(Port: LongInt) : Byte`
`procedure ReadPortB(Port: LongInt; var Buf; Count: LongInt)`

Visibility: default

Description: The procedural form of `ReadPortB` reads `Count` bytes from port `Port` and stores them in `Buf`. There must be enough memory allocated at `Buf` to store `Count` bytes.

The functional form of `ReadPortB` reads 1 byte from port `B` and returns the byte that was read.

Note that you need permission to read a port. This permission can be set by the root user with the `FpIOPerm` (1309) call.

Errors: In case of an error (not enough permissions read this port), runtime 216 (*Access Violation*) will occur.

See also: `FpIOPerm` (1309), `ReadPort` (1310), `ReadPortW` (1311), `ReadPortL` (1311), `WritePort` (1311), `WritePortB` (1312), `WritePortL` (1312), `WritePortW` (1312)

ReadPortL

Synopsis: Read longints from a PC port

Declaration: `function ReadPortL(Port: LongInt) : LongInt`
`procedure ReadPortL(Port: LongInt; var Buf; Count: LongInt)`

Visibility: default

Description: The procedural form of `ReadPortL` reads `Count` longints from port `Port` and stores them in `Buf`. There must be enough memory allocated at `Buf` to store `Count` Longints.

The functional form of `ReadPortL` reads 1 longint from port `B` and returns the longint that was read.

Note that you need permission to read a port. This permission can be set by the root user with the `FpIOPerm` (1309) call.

Errors: In case of an error (not enough permissions read this port), runtime 216 (*Access Violation*) will occur.

See also: `FpIOPerm` (1309), `ReadPort` (1310), `ReadPortW` (1311), `ReadPortB` (1310), `WritePort` (1311), `WritePortB` (1312), `WritePortL` (1312), `WritePortW` (1312)

ReadPortW

Synopsis: Read Words from a PC port

Declaration: `function ReadPortW(Port: LongInt) : Word`
`procedure ReadPortW(Port: LongInt; var Buf; Count: LongInt)`

Visibility: default

Description: The procedural form of `ReadPortW` reads `Count` words from port `Port` and stores them in `Buf`. There must be enough memory allocated at `Buf` to store `Count` words.

The functional form of `ReadPortW` reads 1 word from port `B` and returns the word that was read.

Note that you need permission to read a port. This permission can be set by the root user with the `FpIOPerm` (1309) call.

Errors: In case of an error (not enough permissions read this port), runtime 216 (*Access Violation*) will occur.

See also: `FpIOPerm` (1309), `ReadPort` (1310), `ReadPortB` (1310), `ReadPortL` (1311), `WritePort` (1311), `WritePortB` (1312), `WritePortL` (1312), `WritePortW` (1312)

WritePort

Synopsis: Write data to PC port

Declaration: `procedure WritePort(Port: LongInt; Value: Byte)`
`procedure WritePort(Port: LongInt; Value: LongInt)`
`procedure WritePort(Port: LongInt; Value: Word)`

Visibility: default

Description: `WritePort` writes `Value` – 1 byte, `Word` or longint – to port `Port`.

Remark: You need permission to write to a port. This permission can be set with root permission with the `FpIOPerm` (1309) call.

Errors: In case of an error (not enough permissions to write to this port), runtime 216 (*Access Violation*) will occur.

See also: [FpIOPerm \(1309\)](#), [WritePortB \(1312\)](#), [WritePortL \(1312\)](#), [WritePortW \(1312\)](#), [ReadPortB \(1310\)](#), [ReadPortL \(1311\)](#), [ReadPortW \(1311\)](#)

WritePortB

Synopsis: Write byte to PC port

Declaration: `procedure WritePortB(Port: LongInt; Value: Byte)`
`procedure WritePortB(Port: LongInt; var Buf; Count: LongInt)`

Visibility: default

Description: The first form of `WritePortB` writes 1 byte to port `Port`. The second form writes `Count` bytes from `Buf` to port `Port`.

Remark: You need permission to write to a port. This permission can be set with root permission with the [FpIOPerm \(1309\)](#) call.

Errors: In case of an error (not enough permissions to write to this port), runtime 216 (*Access Violation*) will occur.

See also: [FpIOPerm \(1309\)](#), [WritePort \(1311\)](#), [WritePortL \(1312\)](#), [WritePortW \(1312\)](#), [ReadPortB \(1310\)](#), [ReadPortL \(1311\)](#), [ReadPortW \(1311\)](#)

WritePortL

Synopsis: Write longint to PC port.

Declaration: `procedure WritePortL(Port: LongInt; Value: LongInt)`
`procedure WritePortL(Port: LongInt; var Buf; Count: LongInt)`

Visibility: default

Description: The first form of `WritePortB` writes 1 byte to port `Port`. The second form writes `Count` bytes from `Buf` to port `Port`.

Remark: You need permission to write to a port. This permission can be set with root permission with the [FpIOPerm \(1309\)](#) call.

Errors: In case of an error (not enough permissions to write to this port), runtime 216 (*Access Violation*) will occur.

See also: [FpIOPerm \(1309\)](#), [WritePort \(1311\)](#), [WritePortB \(1312\)](#), [WritePortW \(1312\)](#), [ReadPortB \(1310\)](#), [ReadPortL \(1311\)](#), [ReadPortW \(1311\)](#)

WritePortW

Synopsis: Write Word to PC port

Declaration: `procedure WritePortW(Port: LongInt; Value: Word)`
`procedure WritePortW(Port: LongInt; var Buf; Count: LongInt)`

Visibility: default

Description: The first form of `WritePortB` writes 1 byte to port `Port`. The second form writes `Count` bytes from `Buf` to port `Port`.

Remark: You need permission to write to a port. This permission can be set with root permission with the `FpIOPerm` (1309) call.

Errors: In case of an error (not enough permissions to write to this port), runtime 216 (*Access Violation*) will occur.

See also: `FpIOPerm` (1309), `WritePort` (1311), `WritePortL` (1312), `WritePortB` (1312), `ReadPortB` (1310), `ReadPortL` (1311), `ReadPortW` (1311)